CS 371: Introduction to Artificial Intelligence

Game-Tree Search

Gettysburg College
Computer Science

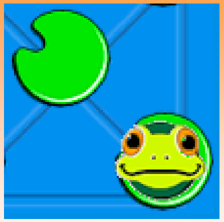
Game-Playing

- Introduction
- Minimax
- Alpha-beta pruning
- Expectiminimax

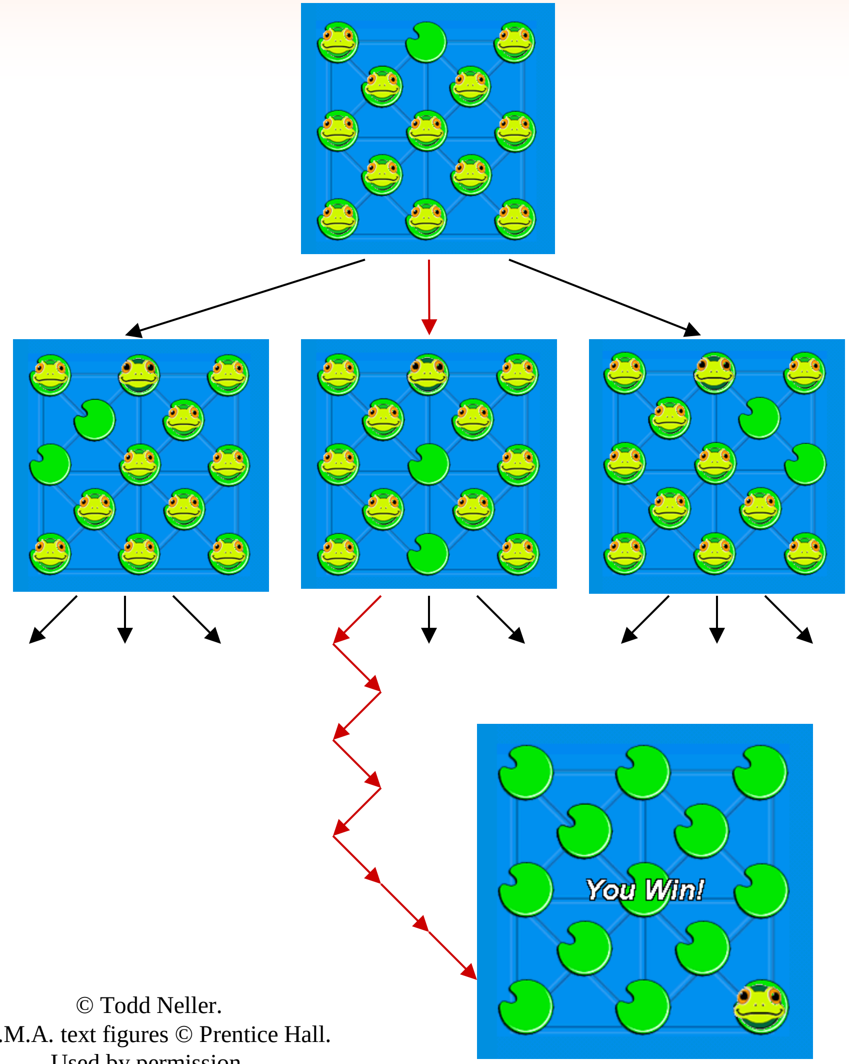
Games as Search Problems

- Previously, we've looked at search problems with *static environments*. (One agent affects the environment.)
- Now, we generalize just a bit and allow two agents to affect the environment in turn. → *dynamic environment*
- Previously, we've looked for a sequence of actions to a goal state.
- Now, we're looking for a sequence of actions which maximizes some utility measure regardless of how an adversarial agent acts.

Example of Tree Search



- Search: Peg solitaire
  - jump a peg over another to empty space, removing jumped peg
  - Initial state: only one space empty
  - Goal state: only one space occupied
  - Find sequence of jumps from initial state to goal

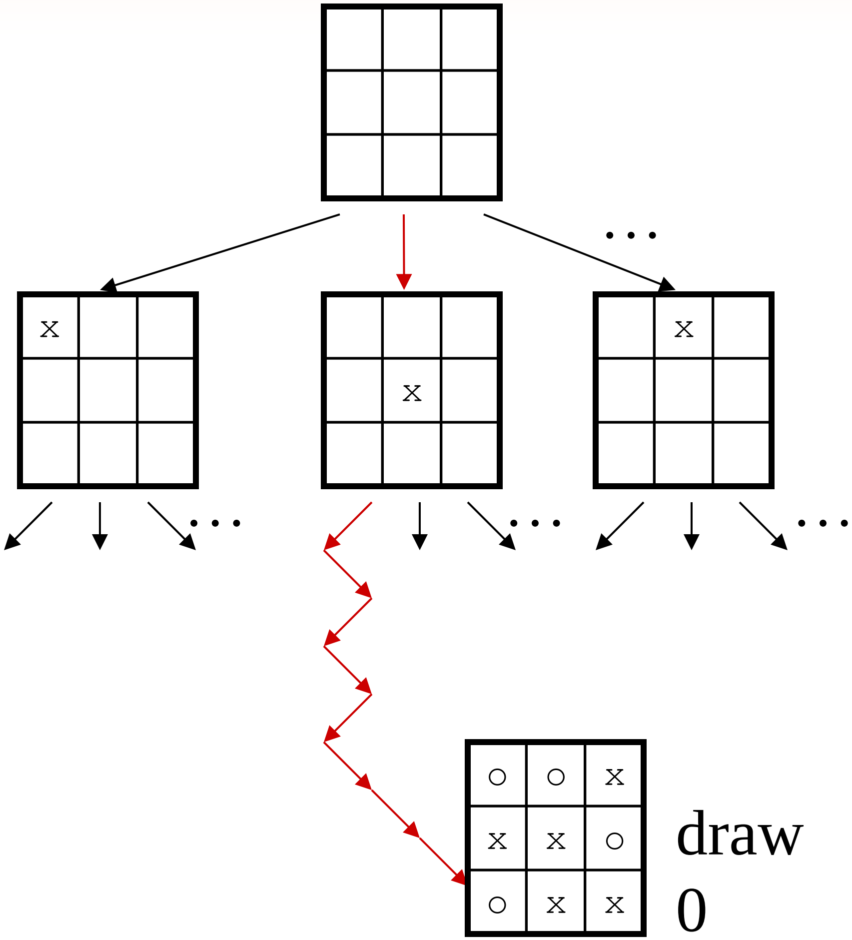


graphics from
hoppers.puzzles.com

Gettysburg College Computer Science

Example of Game-Tree Search

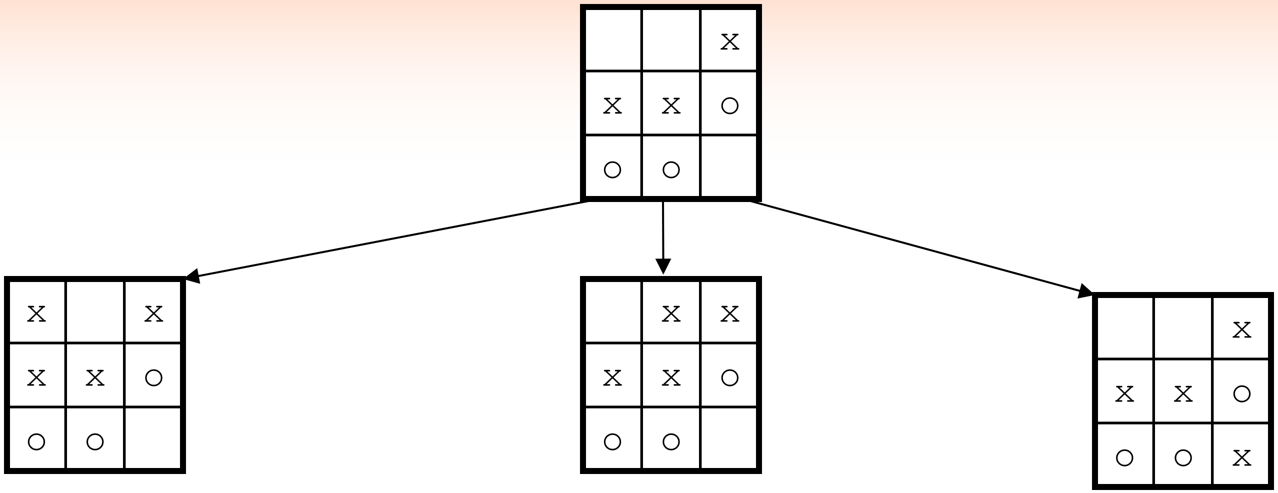
- Search: Tic-Tac-Toe
  - players place X and O in turn.
  - Initial state: empty 3×3 grid
  - Goal state: three of a player's symbol in a row
  - Count win = +1, draw = 0, loss = -1
  - Find sequence of move which maximizes utility regardless of adversarial play

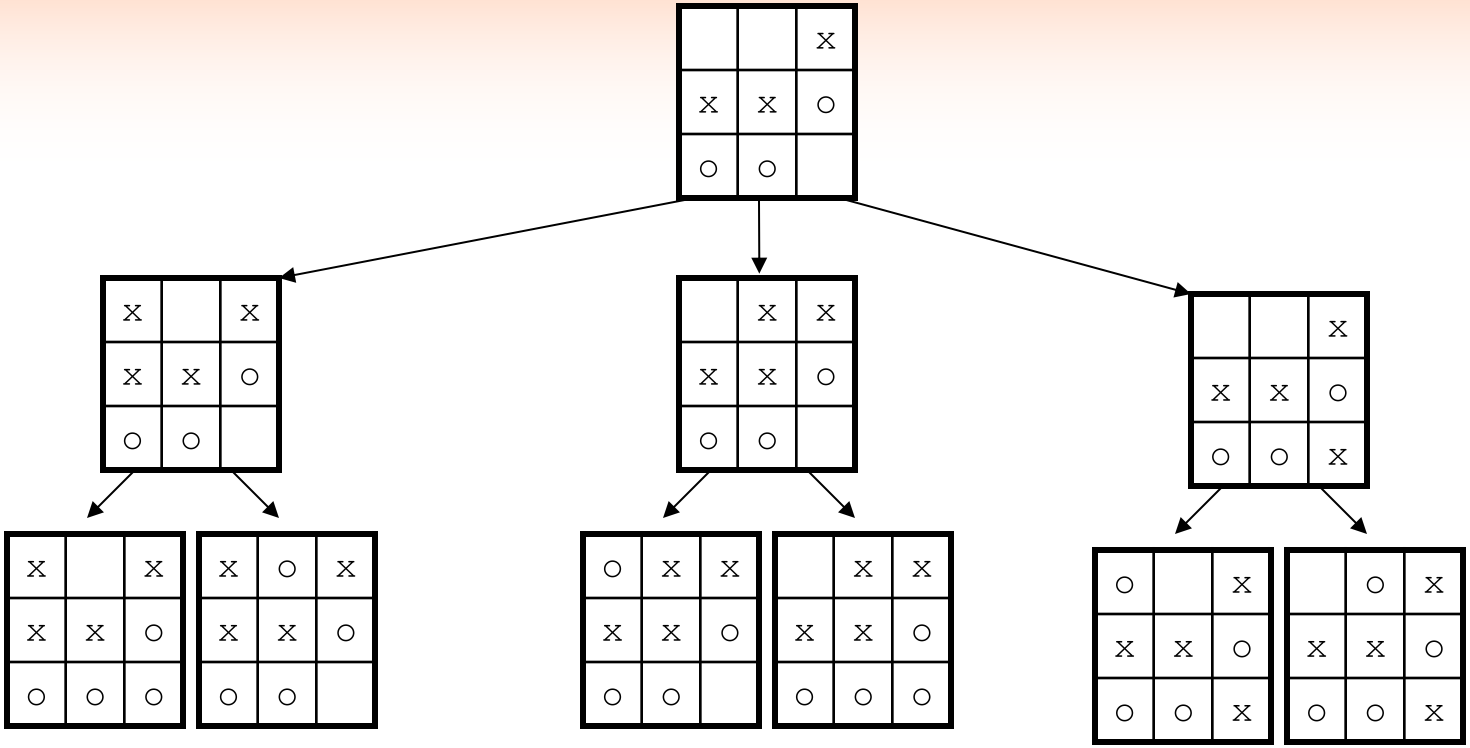


Example of Game-Tree Search

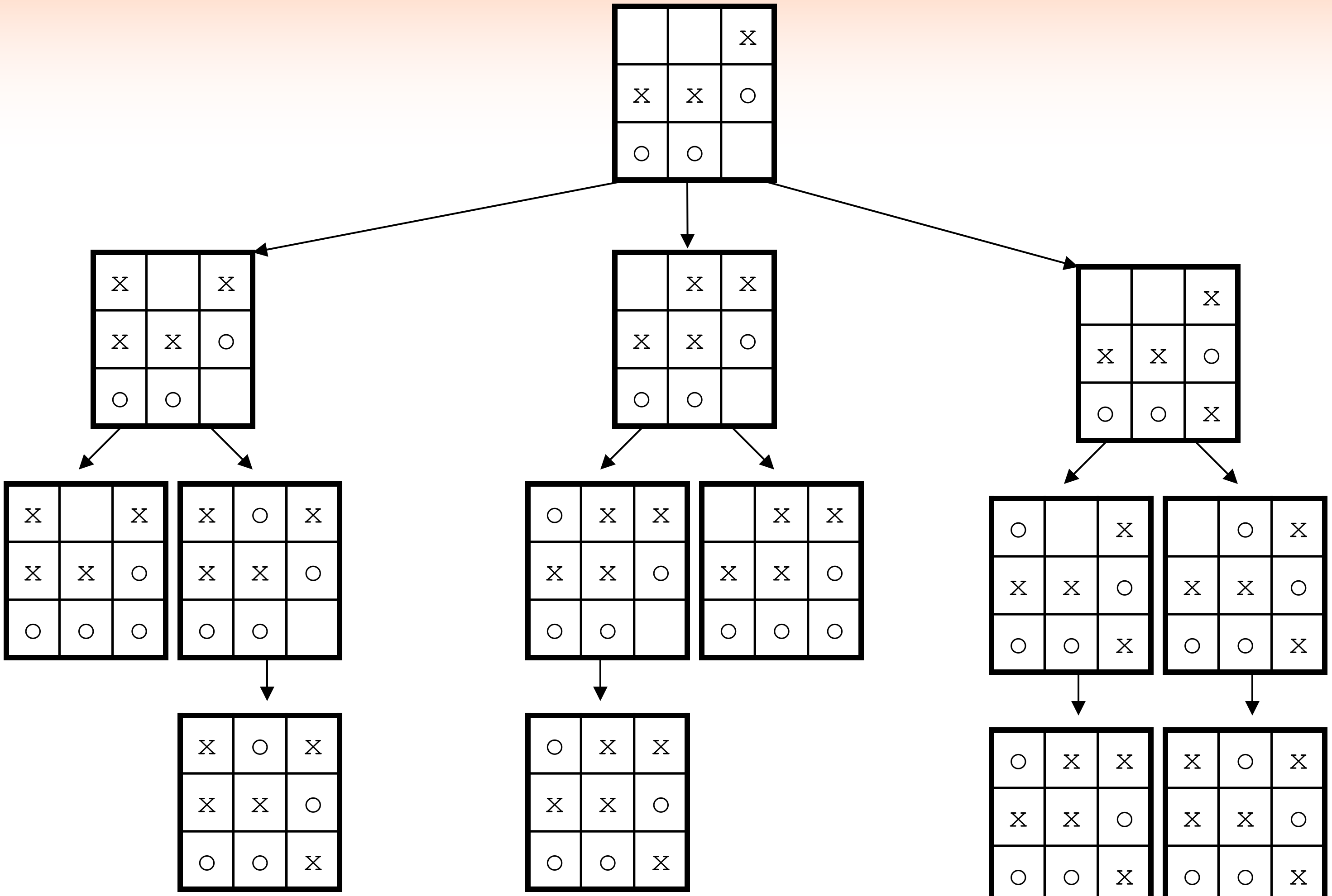
- Suppose you construct the complete tree of possible plays.
- Evaluate terminal states as (+1,0,-1)
- Evaluate non-terminal states as maximum/minimum of children evaluations for player X/O respectively.
- This propagation of evaluations is called *minimax*.
- Consider minimax on a subtree of possible tic-tac-toe plays…

		X
X	X	O
O	O	

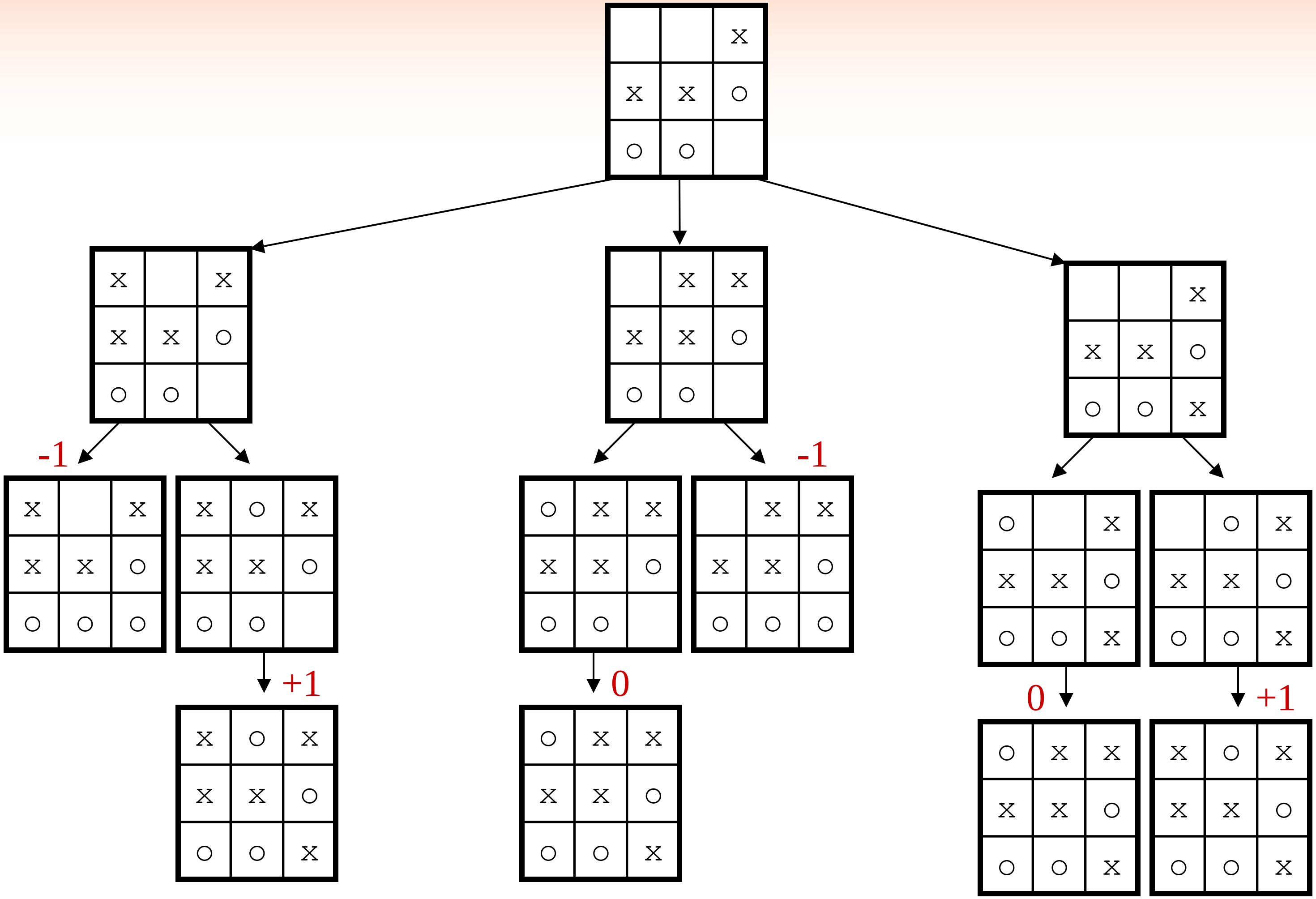

X
X X O
O O
X X
X X O
O O
X X
X X O
O O
X
X X O
O O X



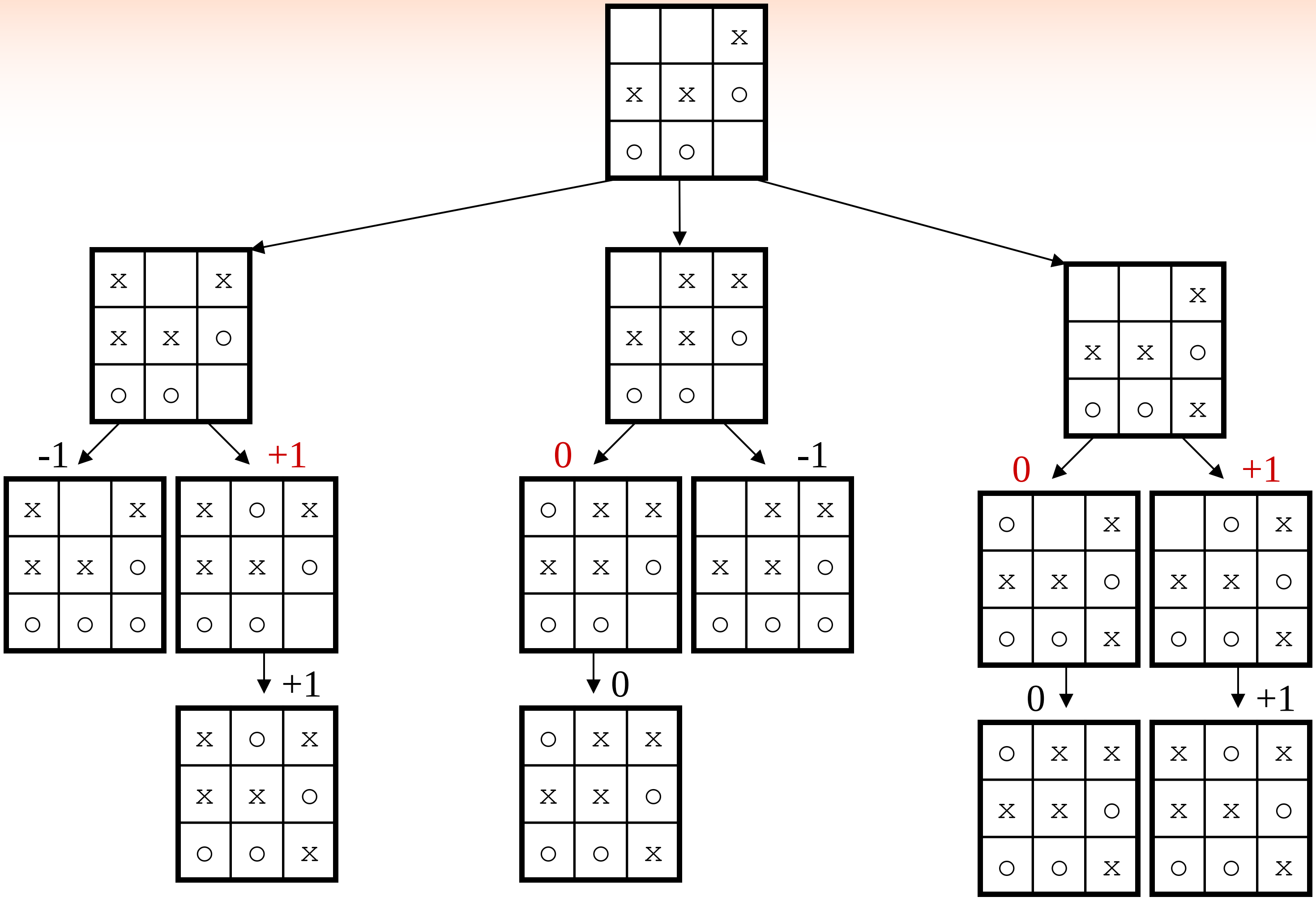
Gettysburg College Computer Science



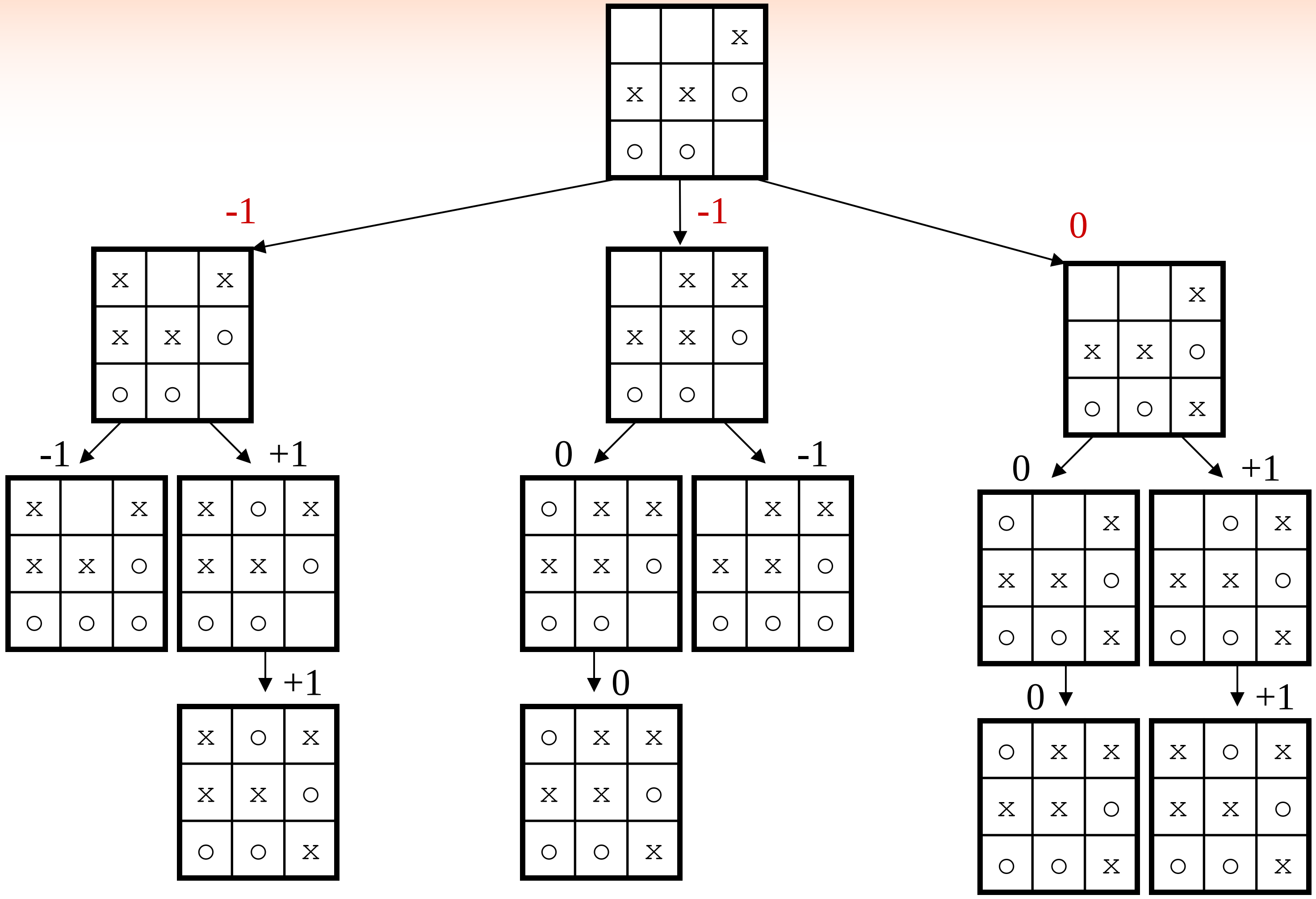
Gettysburg College Computer Science



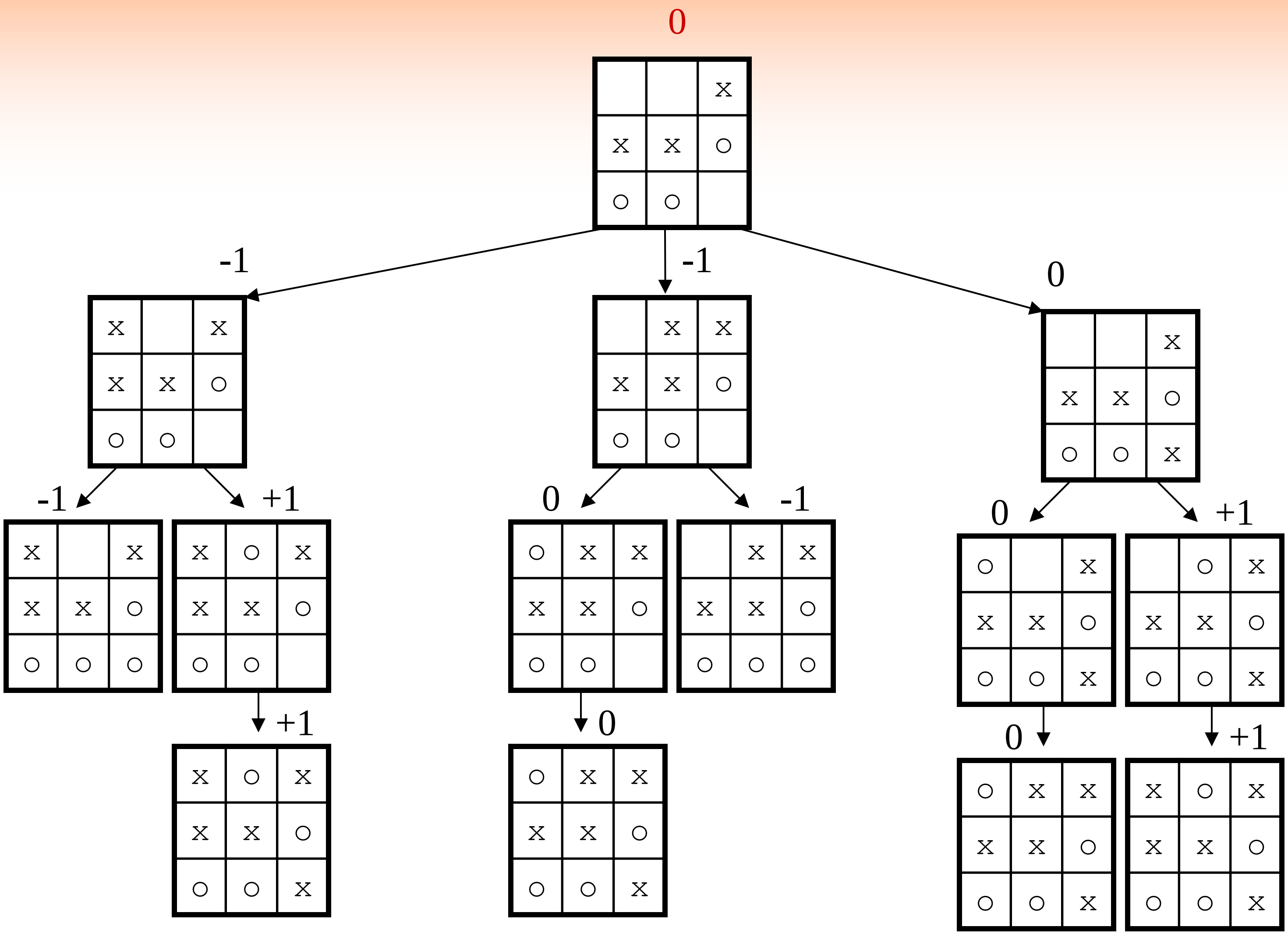
Gettysburg College Computer Science



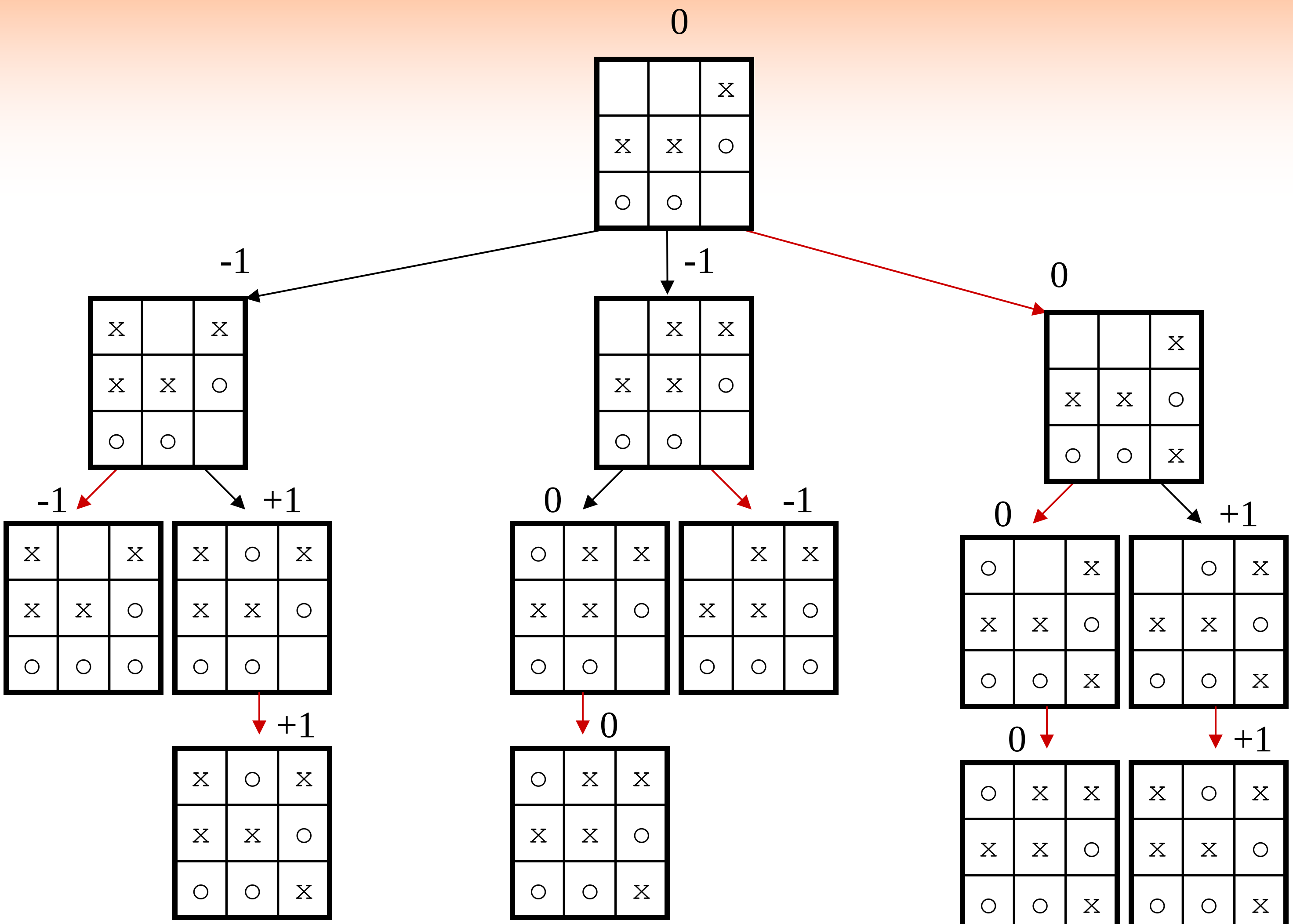
-1
+1
0
-1
0
+1
+1
0
0
+1

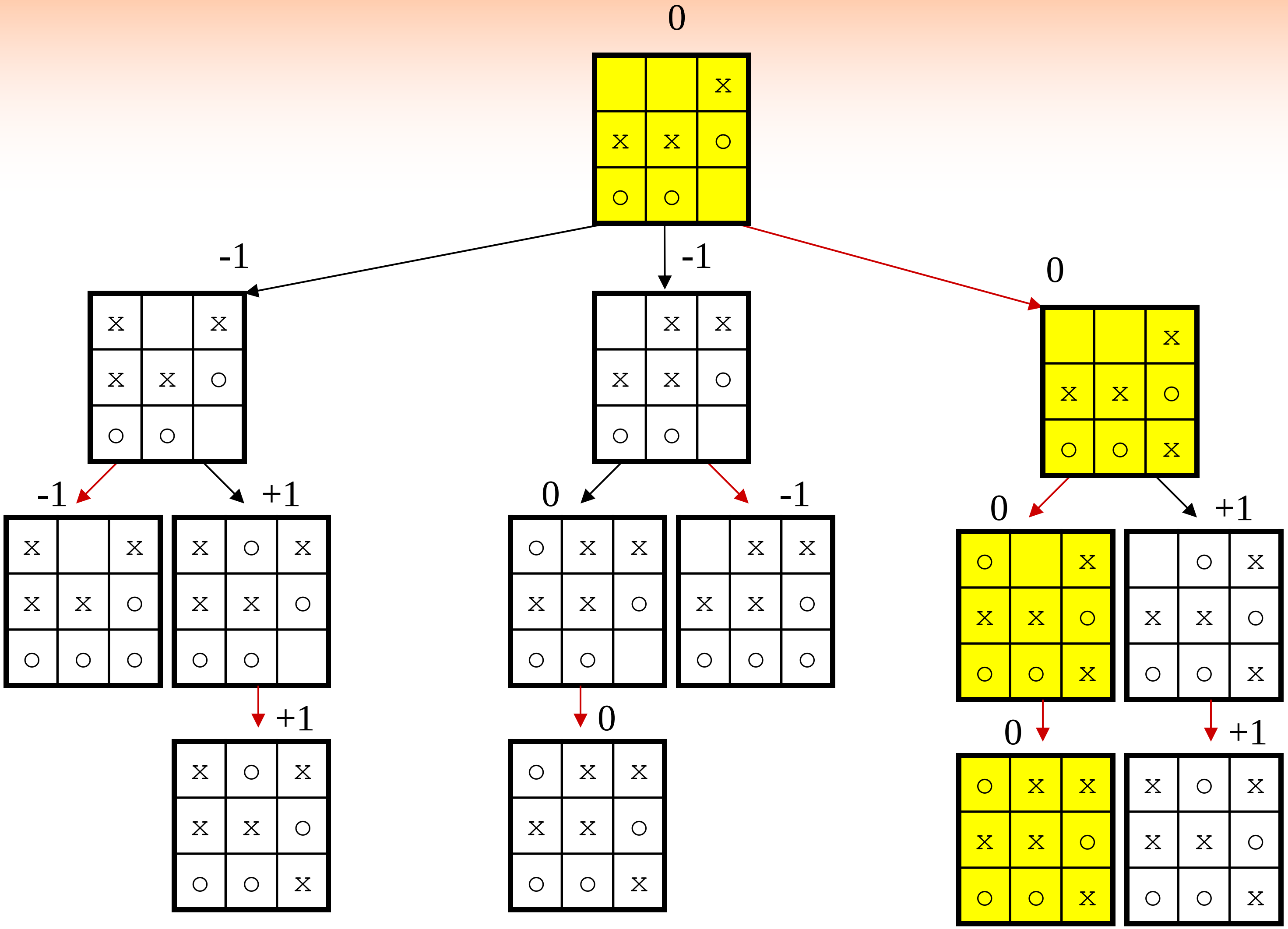


Gettysburg College
Computer Science



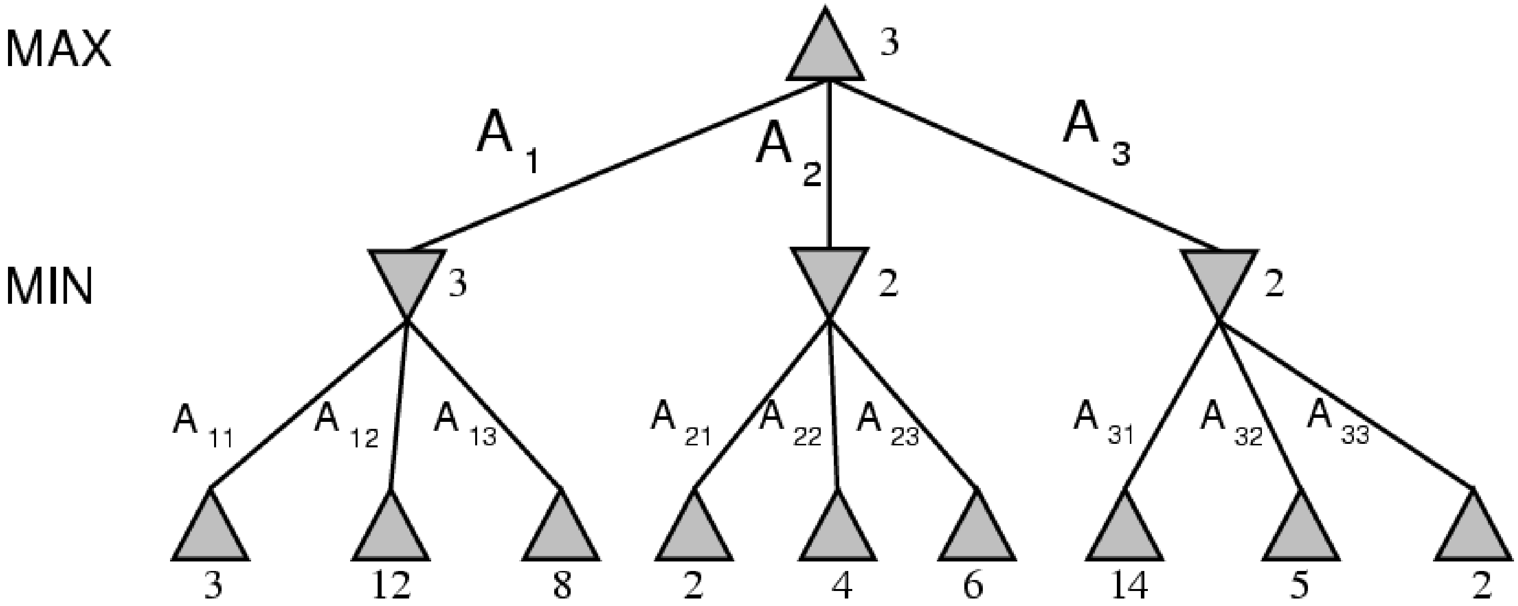
Gettysburg College Computer Science


0
-1
-1
0
-1
+1
0
-1
0
+1
+1
0
0
+1



Gettysburg College Computer Science

Another Minimax Example



Minimax Decision-Making

```
function MINIMAX-DECISION(game) returns an operator

  for each op in OPERATORS[game] do
      VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
  end
  return the op with the highest VALUE[op]
```

```
function MINIMAX-VALUE(state, game) returns a utility value

  if TERMINAL-TEST[game](state) then
      return UTILITY[game](state)
  else if  MAX is to move in state then
      return the highest MINIMAX-VALUE of SUCCESSORS(state)
  else
      return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

Perfect Decisions in Two-Player Games

- Problem definition: initial state, operators, terminal test, utility (or payoff) function
- Given whole game tree, *minimax* yields perfect decisions*
- Minimax: minimum of the maximum of the minimum of the maximum of the…

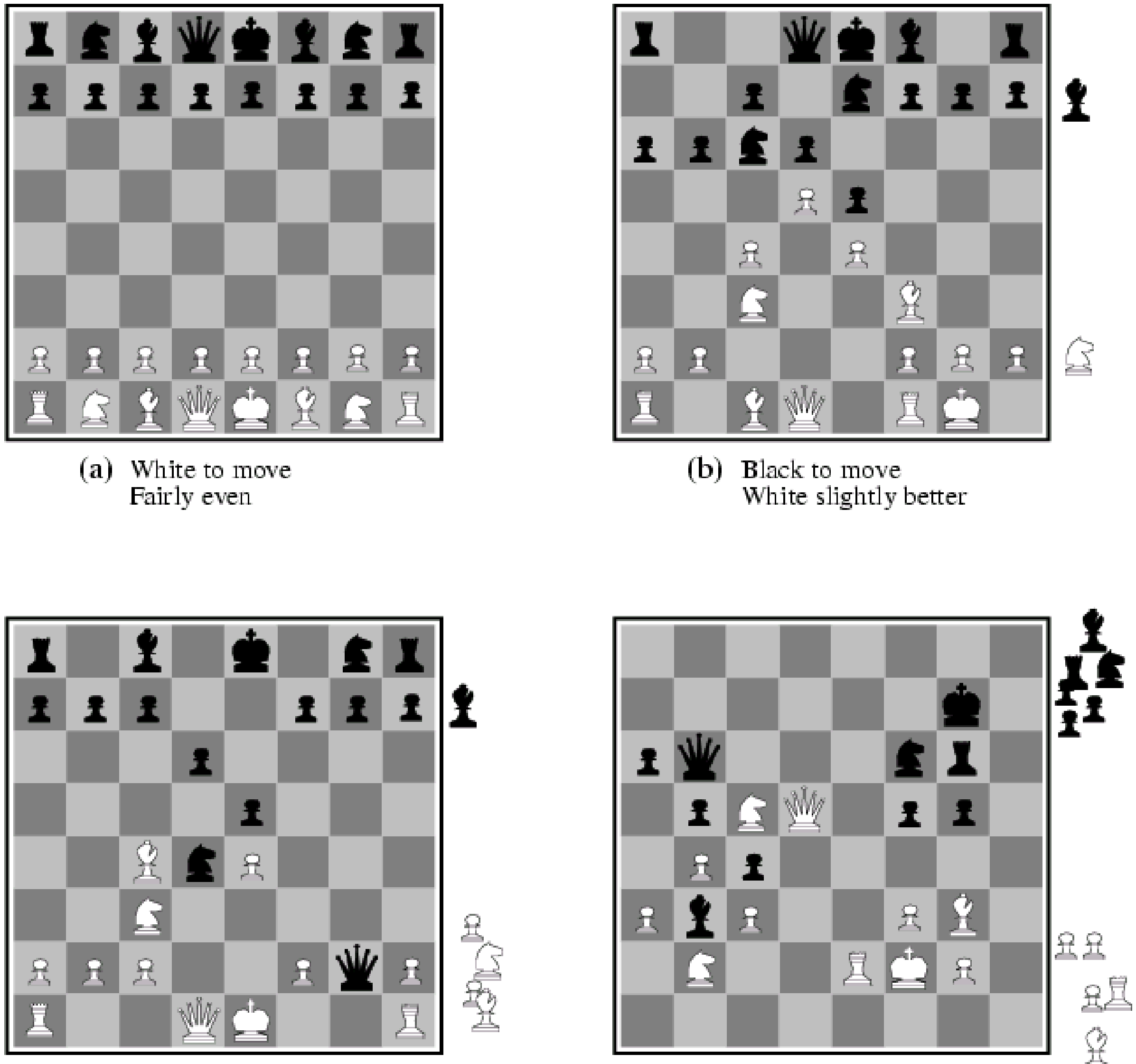
*assuming adversary acts according to minimax → importance of player modeling

- Can't search whole game tree, so…

Imperfect Decisions in Two-Player Games

- Evaluate states passing *cutoff-test* according to *heuristic evaluation function*
- Consider Chess
  - enormous state space
  - can't possibly search whole tree with current computational limitations
- Must
  - limit depth of search
  - evaluate non-terminal nodes at limit

- How would you evaluate these positions?
- Material advantage isn't the whole story.



(a) White to move
Fairly even

(b) Black to move
White slightly better

(c) White to move
Black winning

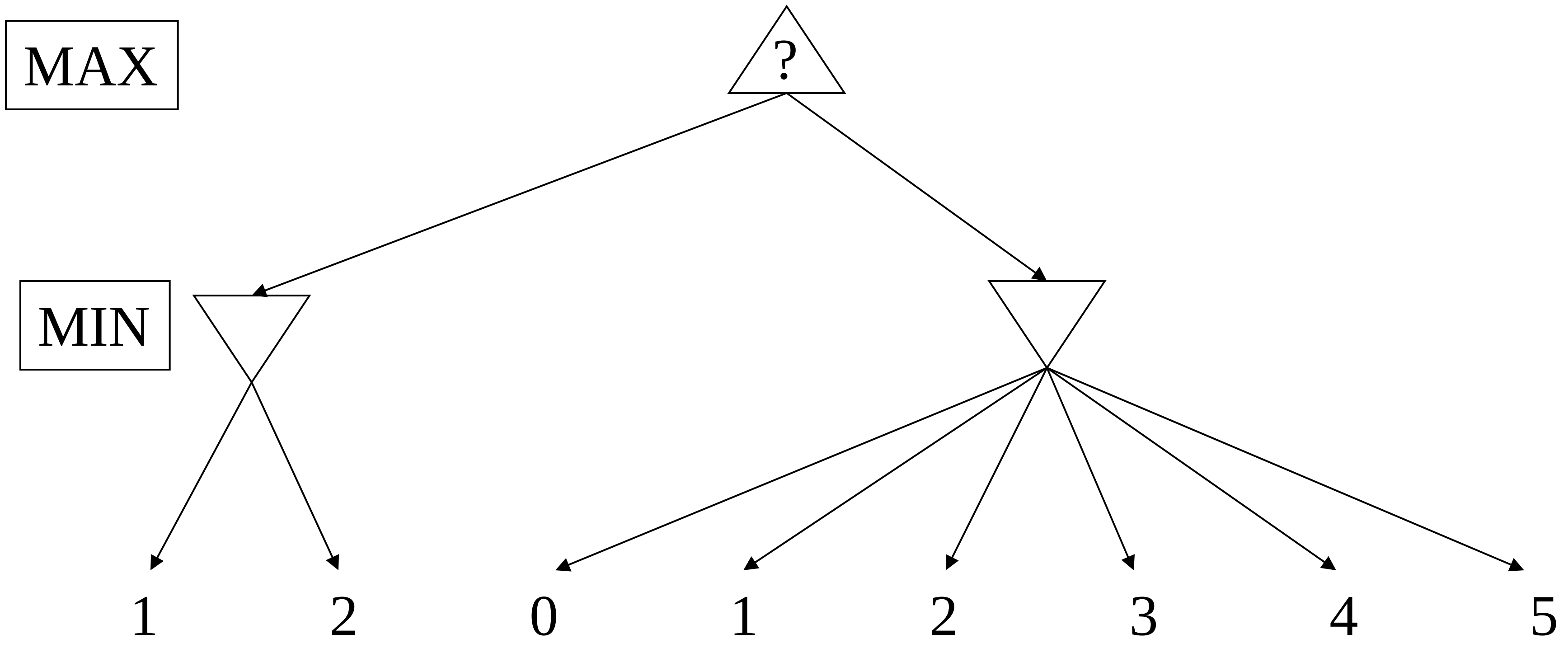
(d) Black to move
White about to lose

Gettysburg College Computer Science

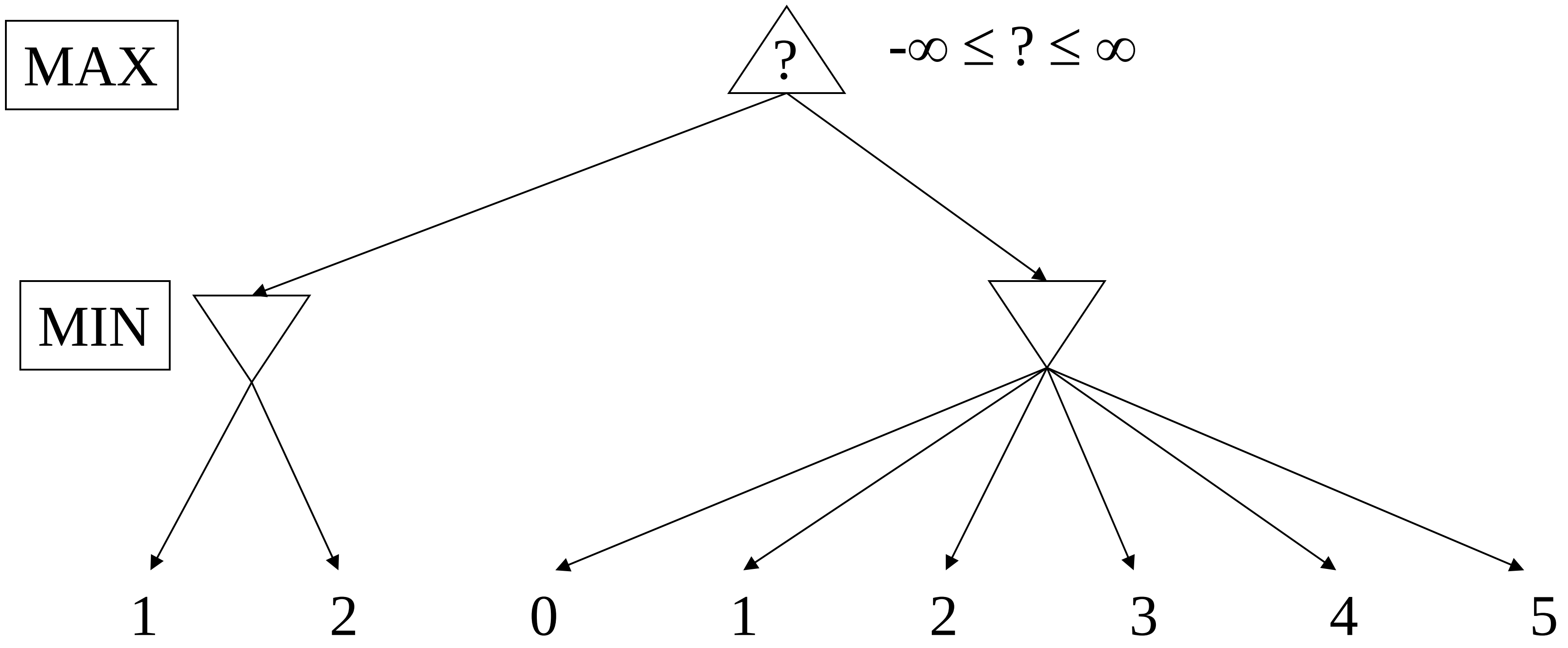
Heuristic Game-Play

- A good evaluation function
  - returns actual value at terminal states,
  - approximates actual value at non-terminal nodes, and
  - isn't too computationally intensive
- Most attribute recent game-playing success to better speed ("brute force") rather than better evaluation (knowledge base)
- Still, most minimax search is pointless…

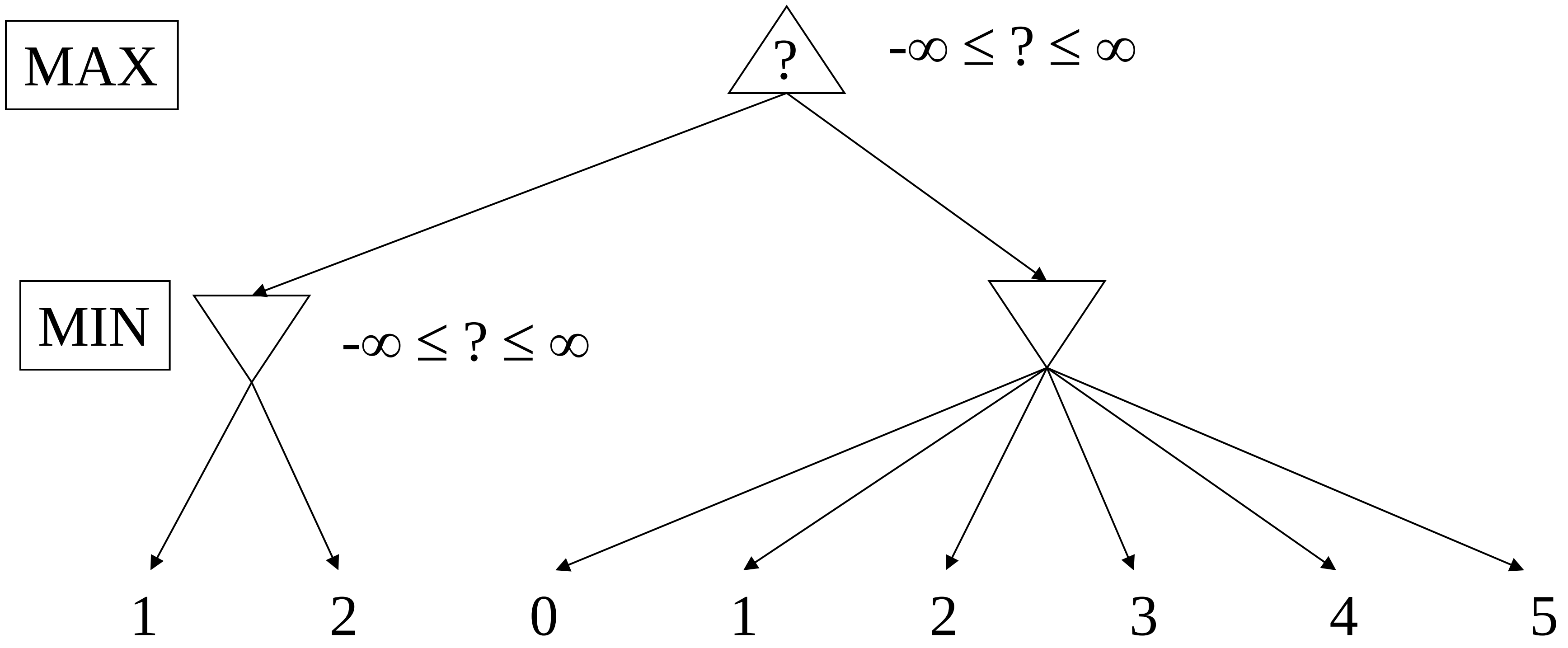
Pruning Example



Pruning Example

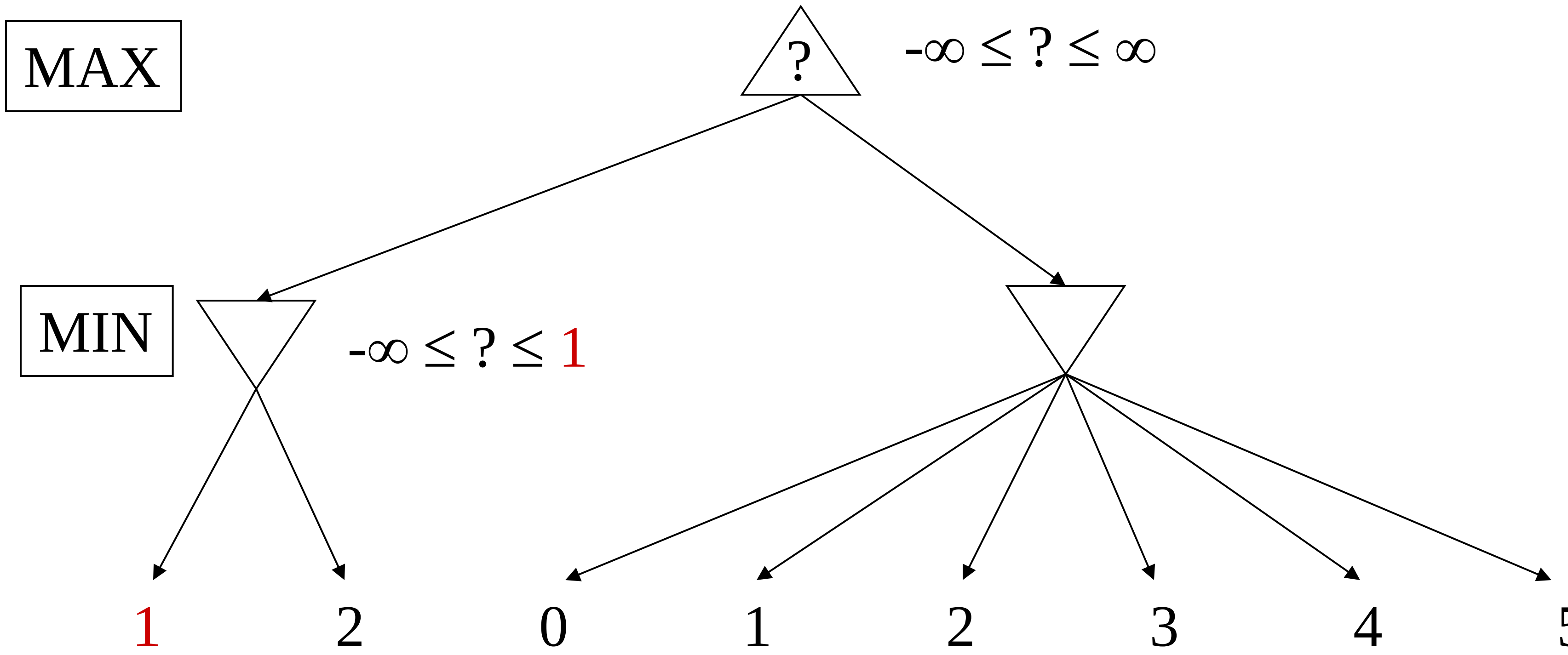


Pruning Example

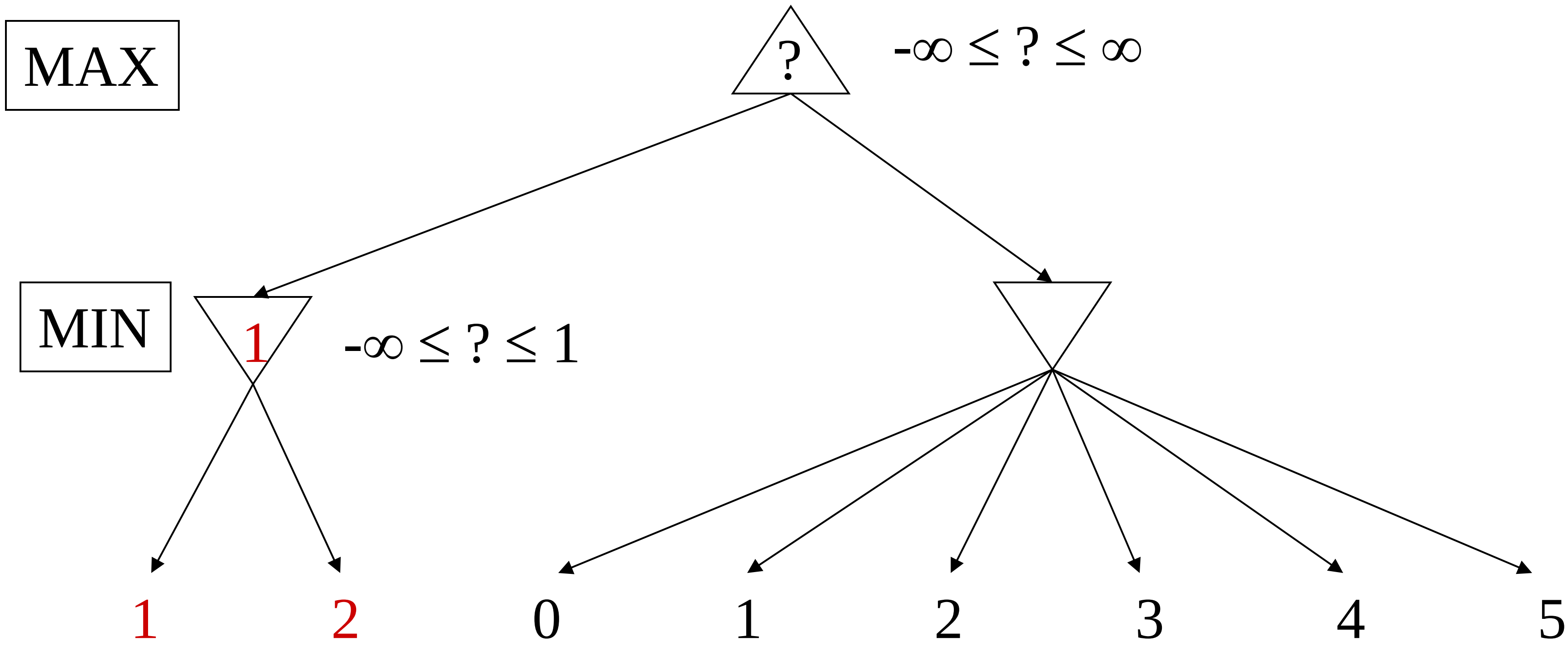


Gettysburg College Computer Science

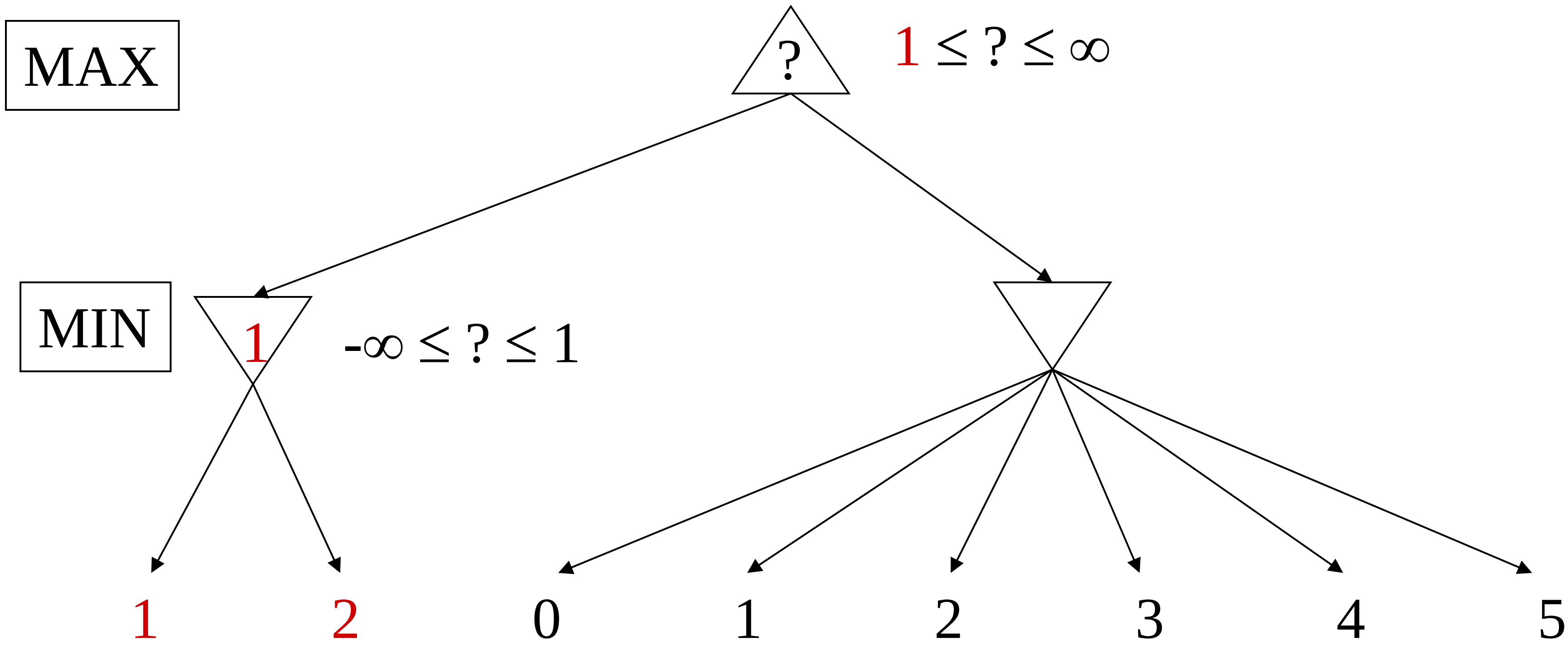
Pruning Example



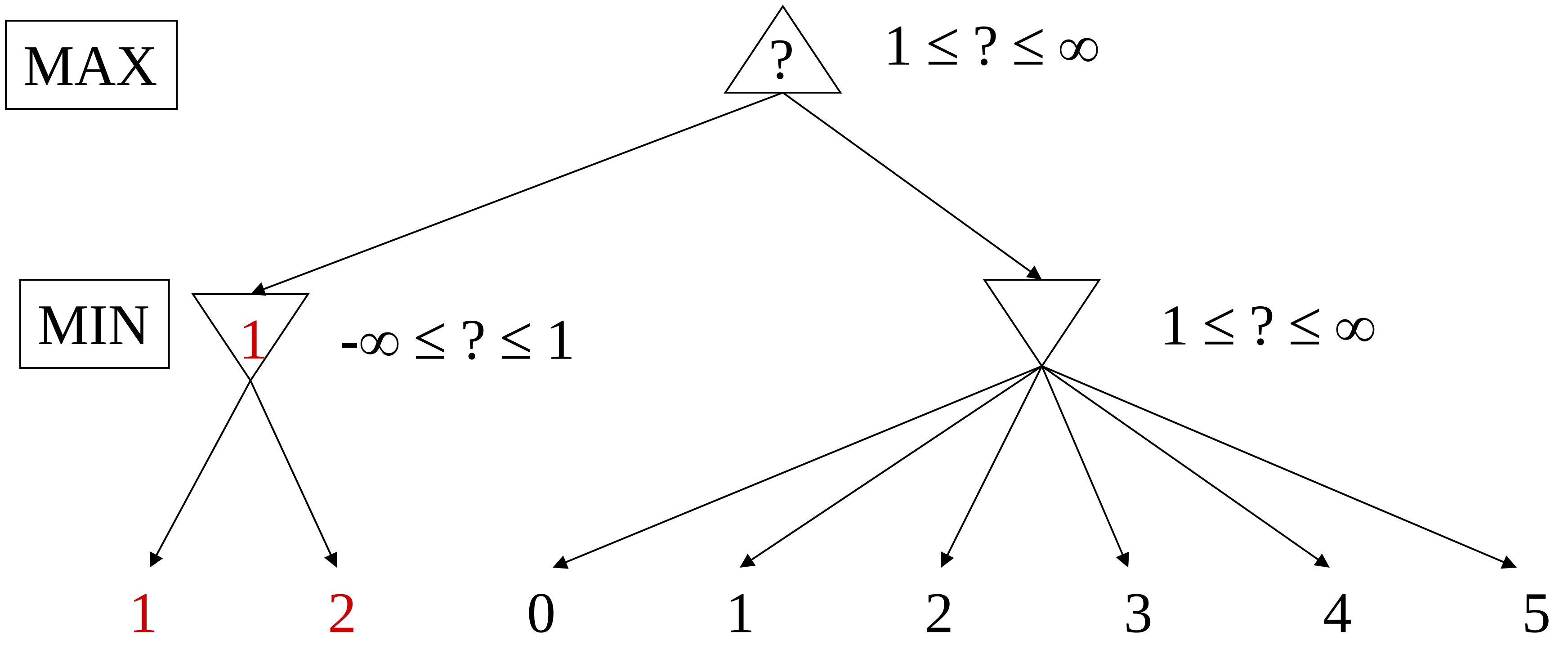
Pruning Example



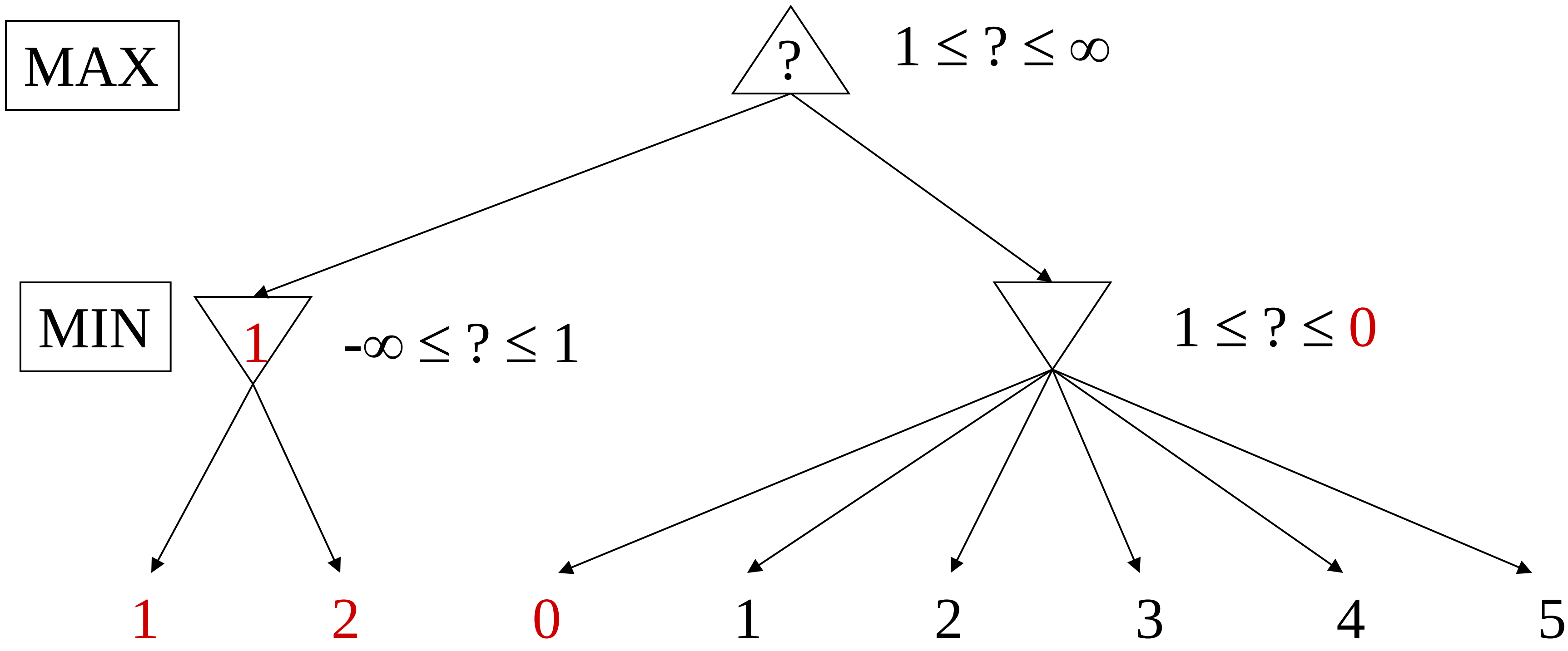
Pruning Example



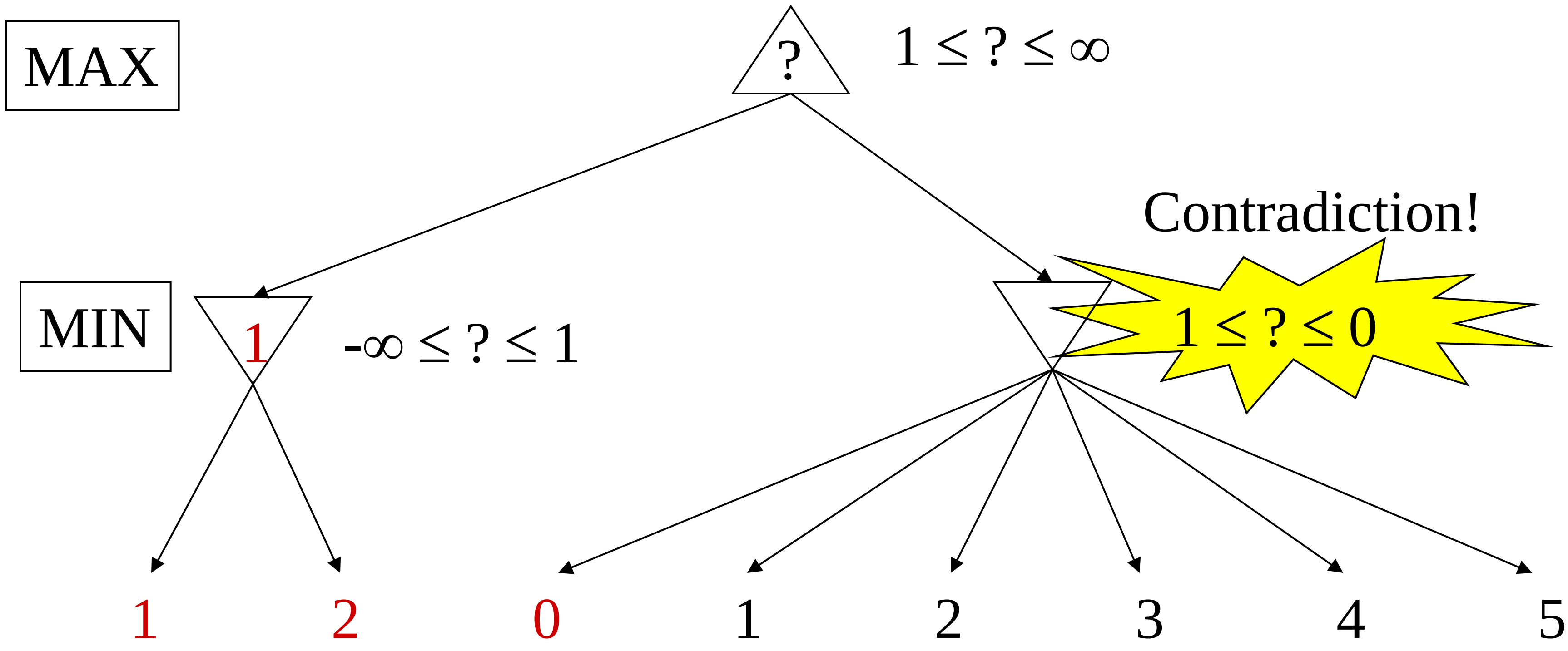
Pruning Example



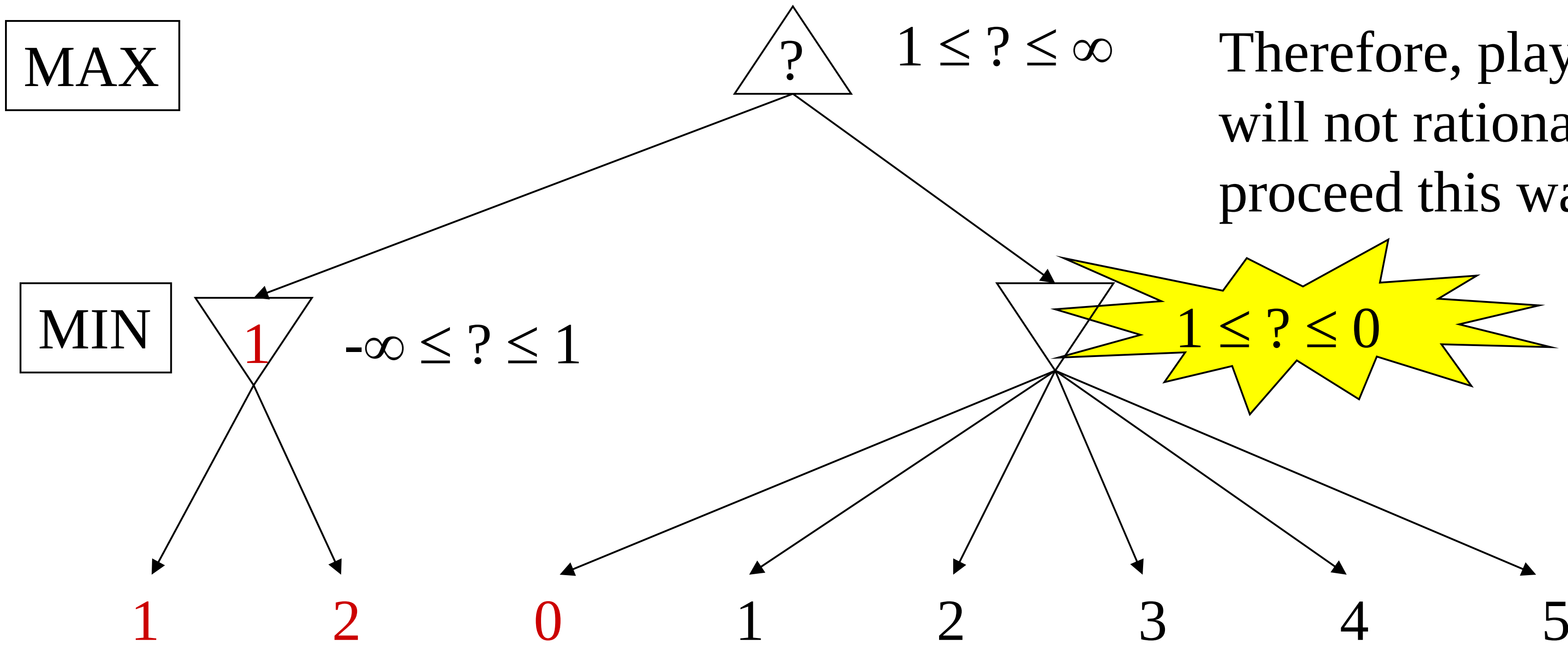
Pruning Example



Pruning Example



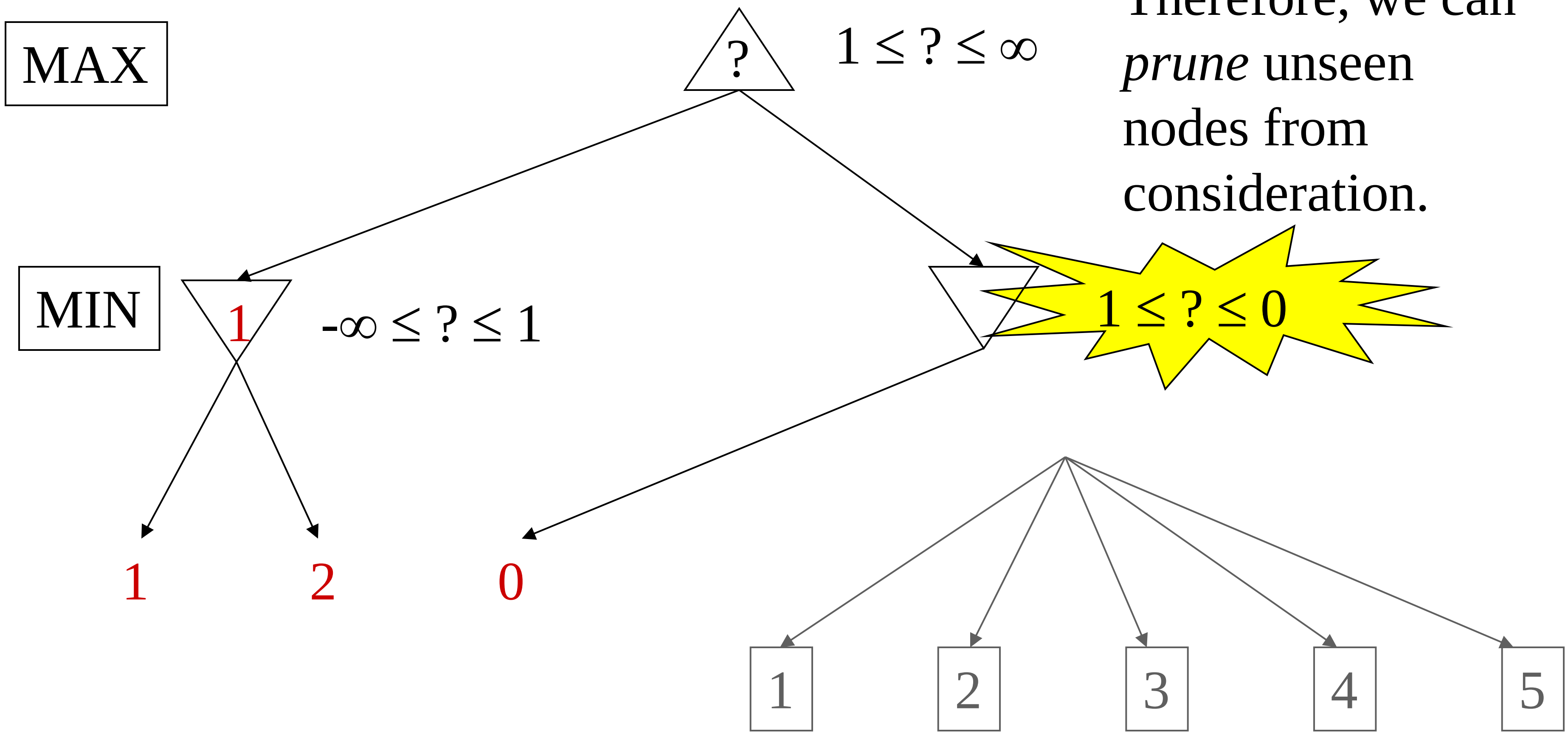
Pruning Example



Therefore, play will not rationally proceed this way

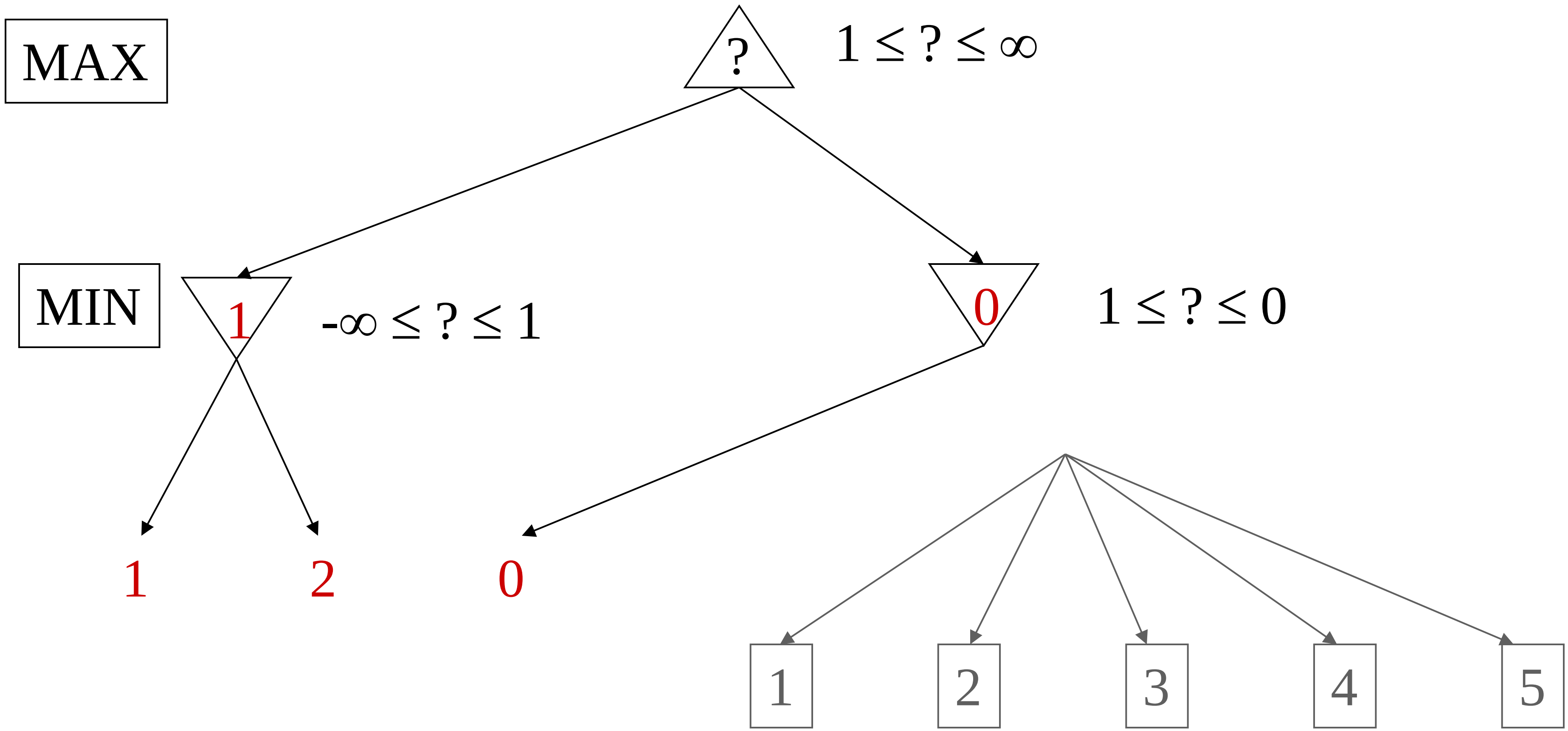
Gettysburg College Computer Science

Pruning Example



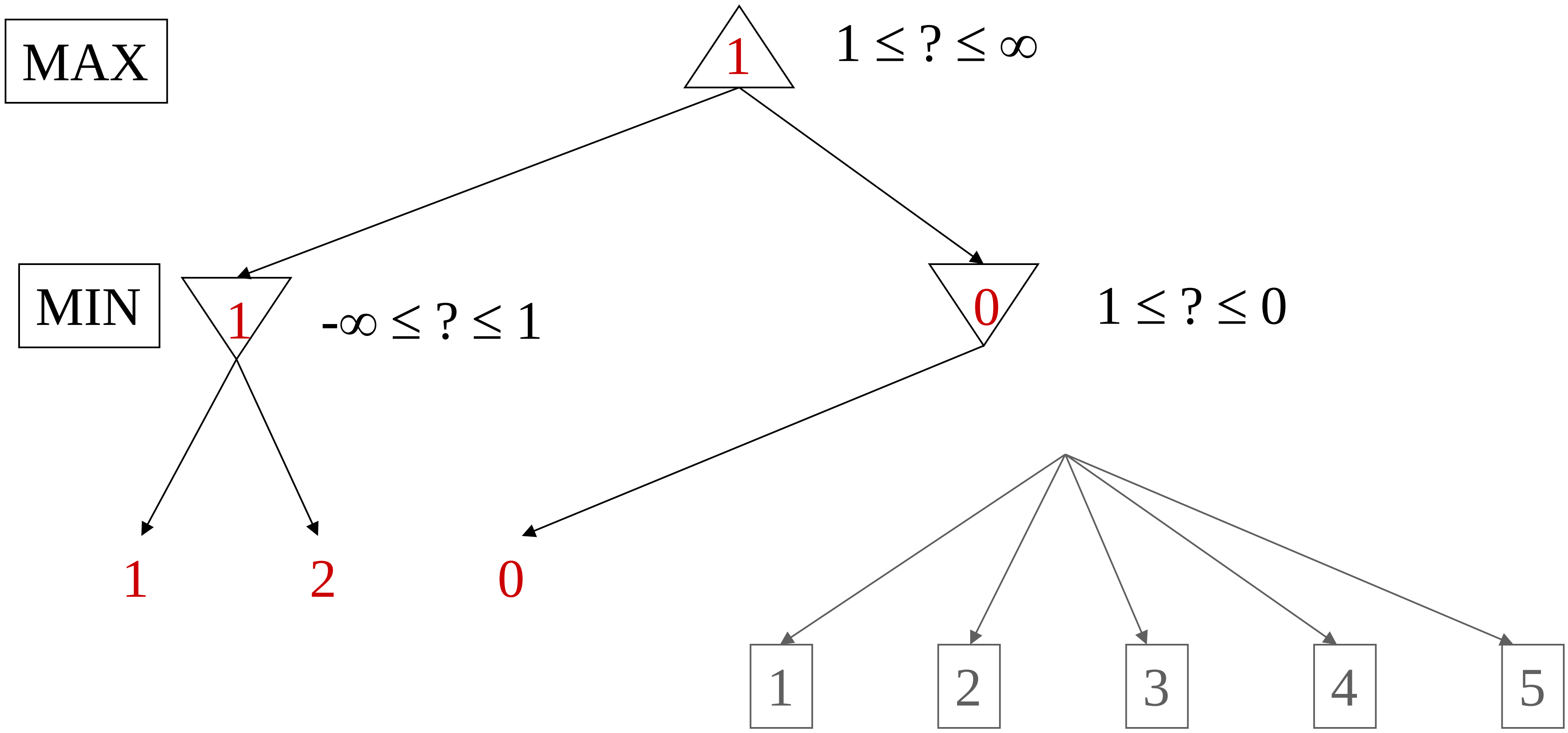
Therefore, we can *prune* unseen nodes from consideration.

Pruning Example



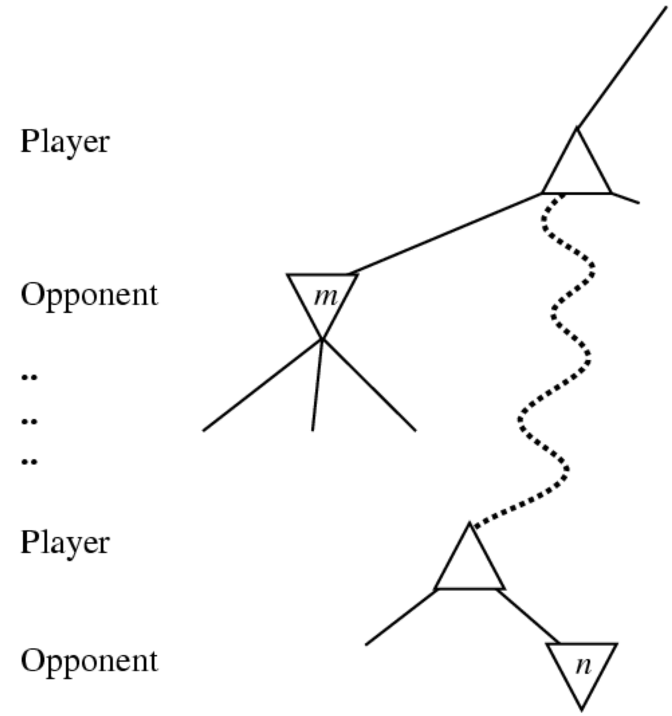
Gettysburg College Computer Science

Pruning Example



Pruning Guarantees

- While we search the tree, we can keep track of guaranteed maximum/minimum utilities if play proceeds to each node.
- When we see a contradiction in guarantees, we can prune remaining children from further consideration, because we've proven a rational player will never reach that node.



Assuming Rationality? Ha!

- What if other player isn't rational?
- If evaluation is perfect, then one can always do as well if not better against an irrational player with rational play.

Alpha-Beta Pruning

- Let α,β be local lower,upper bound guarantees
  - "If play proceeds here, root will score **at least** α."
  - "If play proceeds here, root will score **at most** β."
- Pruning thus according to α and β is called **alpha-beta pruning**.
- Minimax search with alpha-beta pruning is sometimes called **alpha-beta search**.

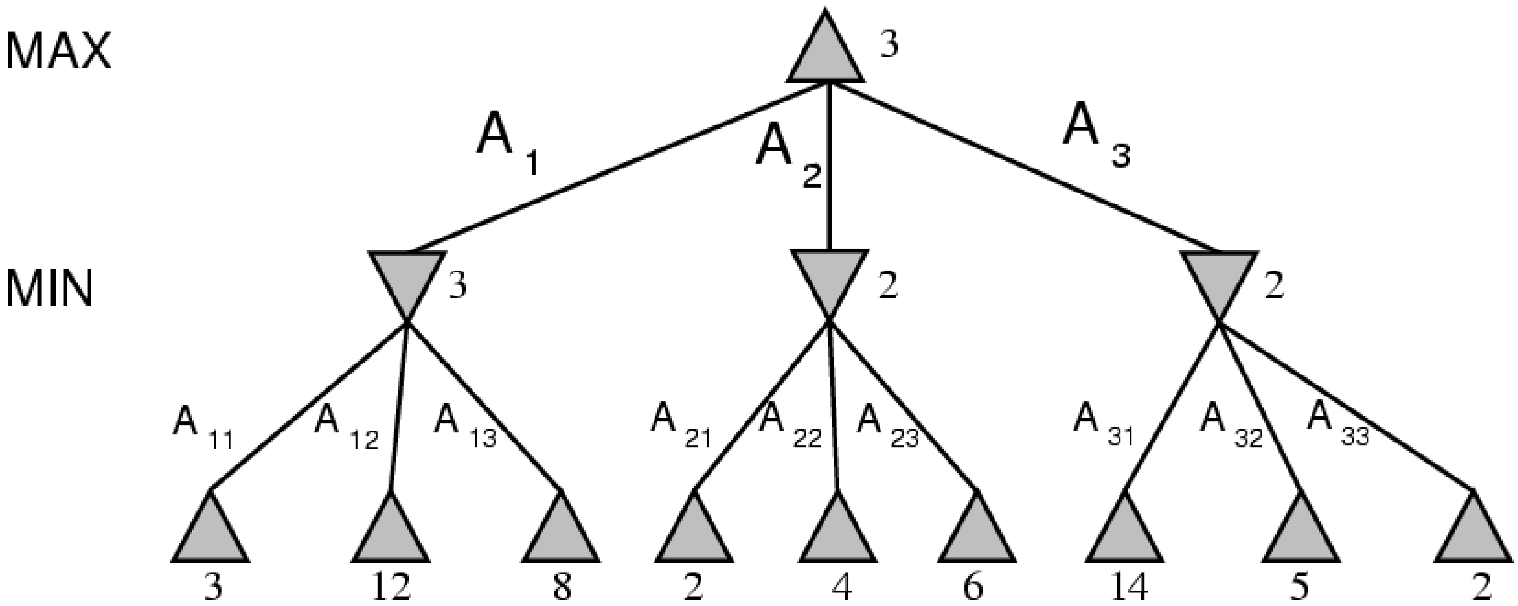
Alpha-Beta Pruning Algorithm

```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
   inputs: state, current state in game
           game, game description
           α, the best score for MAX along the path to state
           β, the best score for MIN along the path to state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
       α ← MAX(α, MIN-VALUE(s, game, α, β))
       if α ≥ β then return β
   end
   return α
```

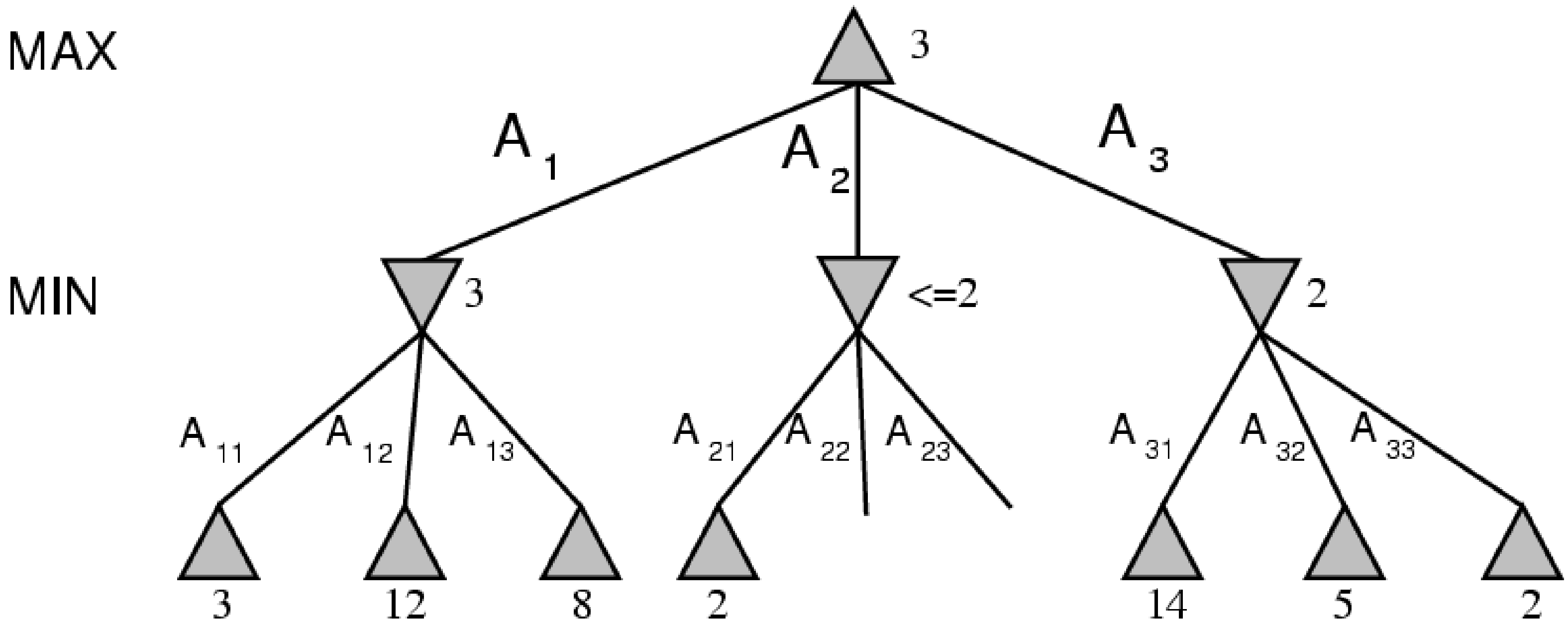
```
function MIN-VALUE(state, game, α, β) returns the minimax value of state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
       β ← MIN(β, MAX-VALUE(s, game, α, β))
       if β ≤ α then return α
   end
   return β
```

Gettysburg College Computer Science

Alpha-Beta Pruning Exercise



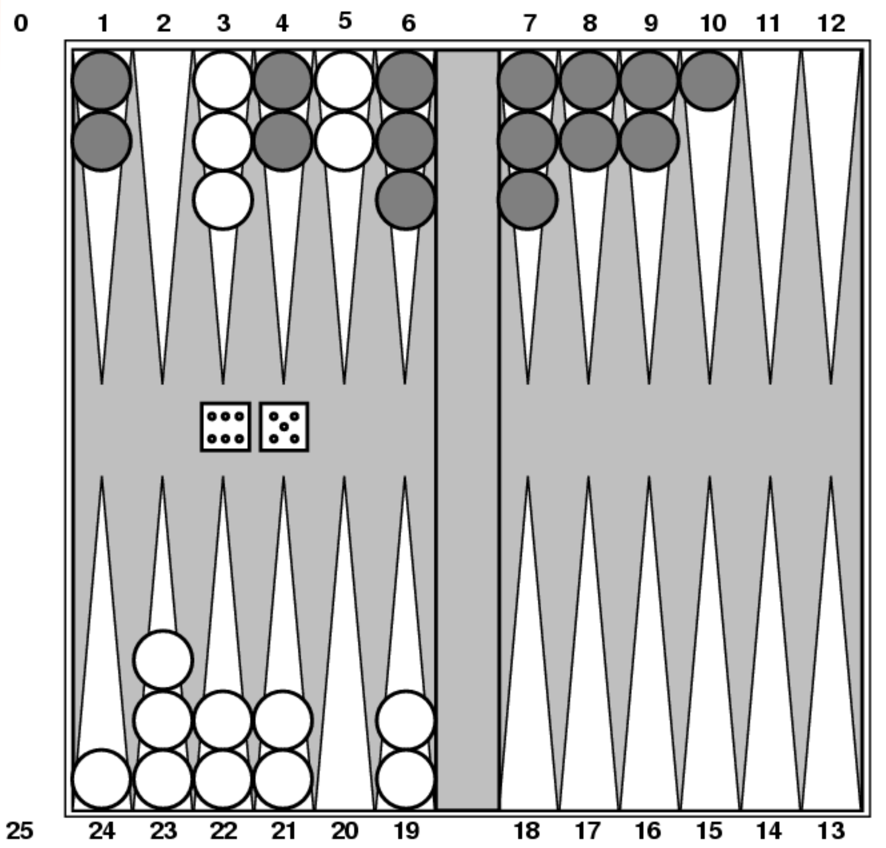
Alpha-Beta Pruning Exercise (cont.)



Gettysburg College Computer Science

Games of Chance

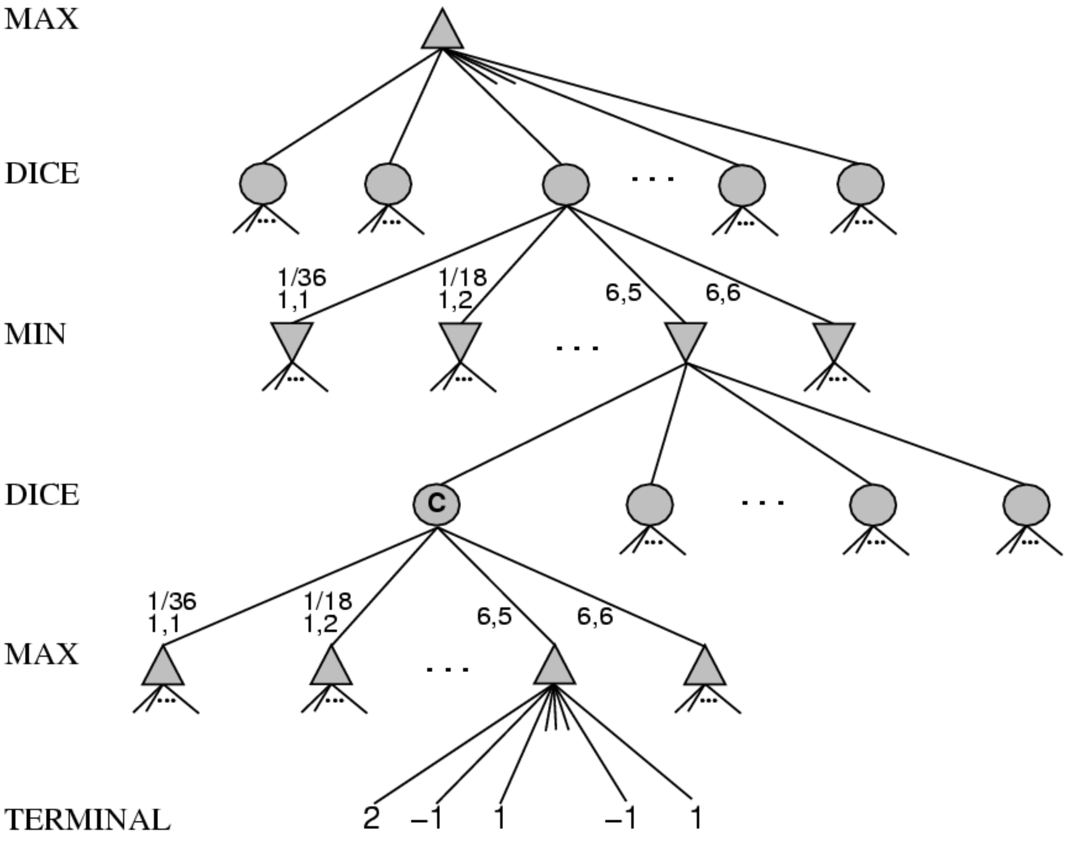
Minimax doesn't help here because dice are random and impartial.



Backgammon

Gettysburg College Computer Science

Chance Nodes

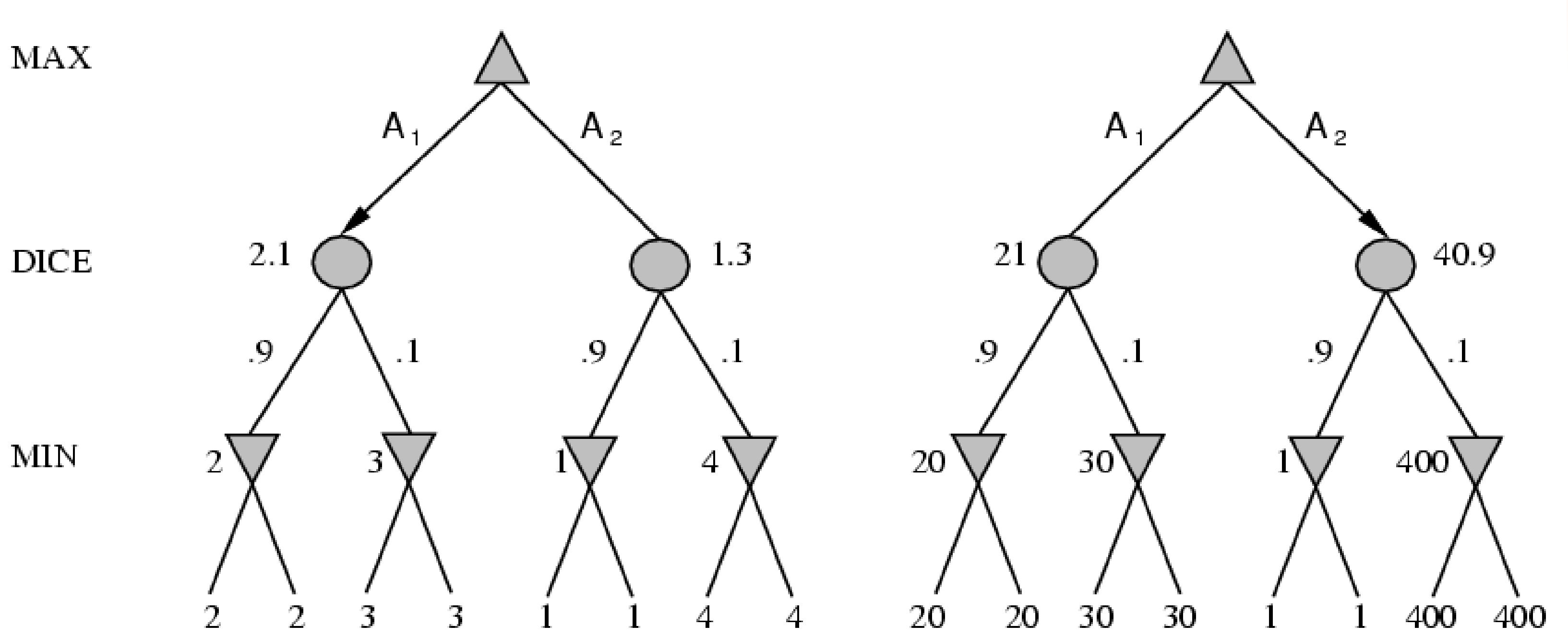


Gettysburg College Computer Science

Expectiminimax

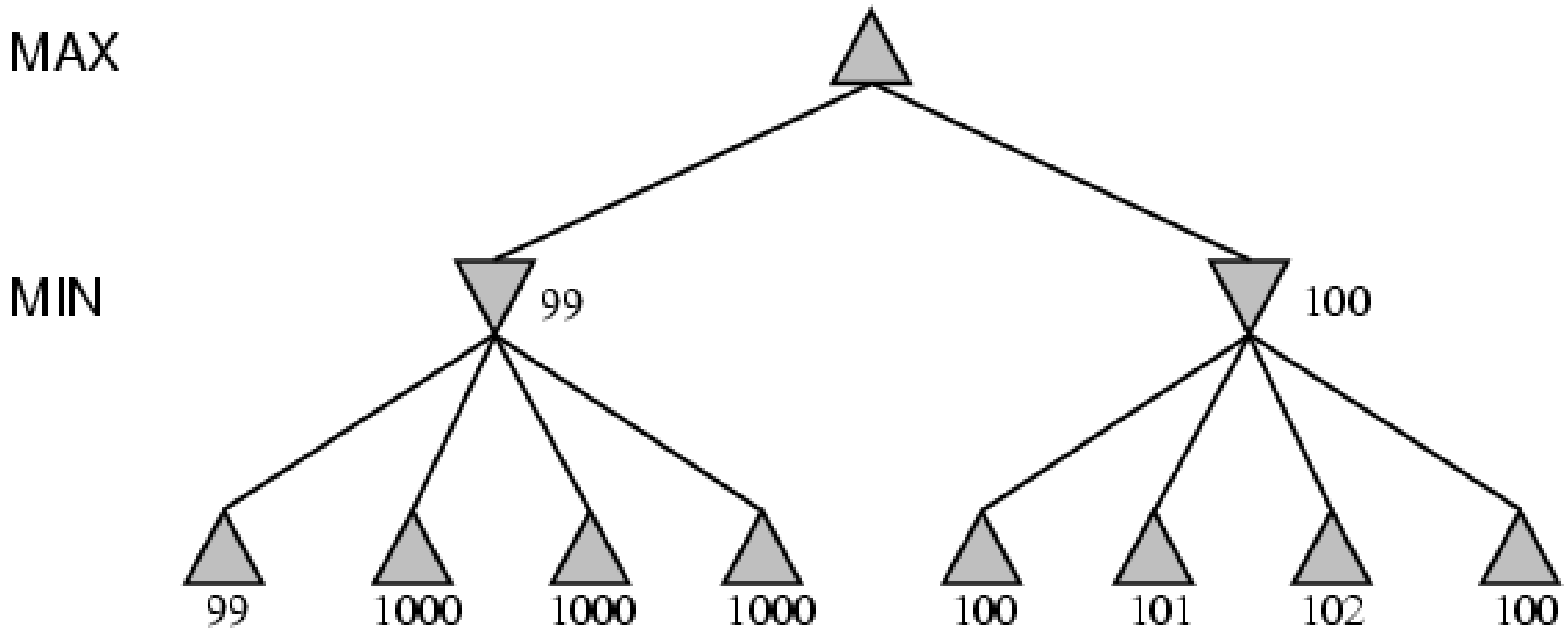
- A chance node is evaluated as follows
  - the value of each child is multiplied times the probability of reaching that child
  - these products are then summed.
- Disadvantages to this approach:
  - branching factor of chance nodes can be large!
  - no pruning allowed
  - evaluation functions are hard!…

Expectiminimax Evaluations



Note that the relative ordering of the leaf values are the same, but the decision has changed! → must approximate *positive linear transformation* of likelihood of winning

Why Minimax Isn't Always Appropriate



What if these numbers are roughly approximate and evaluation occasionally has significant error?